www.farelettronica.com
n. 335/336 – MAGGIO/GIUGNO 2013

Un dispositivo per fotografare i fulmini

Monitoraggio di Arduino via PC

Telecontrollo WI-FI con tablet PC

GOOGLE APP INVENTOR

Crea la tua App Android

Nuovo layout!

- Più facile da navigare
- Più facile da leggere

Acquisizione dati con Raspberry PI

Sommario

335/336 - maggio/giugno 2013

Editoriale

Digitale fruibile

Il passaggio al digitale per una rivista nata in formato cartaceo non è semplice, soprattutto se la rivista è di taglio tecnico, quindi nata per essere letta e "spulciata" riga per riga. E' questo il caso di Fare Elettronica. Il mercato e l'evoluzione dell'editoria ci hanno costretto al passaggio al formato digitale, ma trasformare in pdf una versione cartacea non basta. Ci dobbiamo mettere nei panni dei lettori che si trovano a leggere gli articoli direttamente da un tablet o da un PC e magari stampare quelli più interessanti. Dobbiamo quindi attenerci a poche, semplici, ferree regole per poter rendere la rivista facilmente accessibile e leggibile e, per questo, ci siamo impegnati subito a fondo per cambiare radicalmente la veste grafica ed il formato della rivista. La prima regola è usare un formato orizzontale, che facilita la lettura a schermo intero di un'intera pagina. Nella versione digitale si perde poi il concetto di sequenzialità delle pagine, per cui i contenuti devono essere accessibili da qualsiasi punto della rivista. I pulsanti in alto a destra di ogni pagina consentono di saltare direttamente al sommario o andare avanti e indietro di una pagina. Gli argomenti sono stati suddivisi in tre sezioni: "Progetti", "Tutorial" e "Rubriche".
Si può accedere a ciascuna sezione cliccando sul relativo tab in alto nella pagina. Nella pagina di apertura di ogni articolo sono infine riportati i titoli degli altri articoli della medesima sezione, per potervi accedere immediatamente con un click. Inutile dire che la rivista è ricca di link che, oltre a permettervi di navigare nelle pagine, vi portano anche a contenuti esterni, approfondimenti e documentazione tecnica aggiuntiva. Anche la grafica è stata notevolmente alleggerita sia per facilitare la leggibilità, sia per consentirvi di stampare ciò che vi interessa senza sprecare quintali di inchiostro. A proposito di stampa: se volete stampare le pagine usate le impostazioni per stampare su A4 orizzontale con adattamento alla carta. Il risultato sarà ottimo, vedrete! Siamo convinti che questo sia un buon inizio, ma siamo anche coscienti che tutto è migliorabile, per cui vi invito ad inviare al mio indirizzo di posta elettronica (m.delcorso@inware.it – basta un click!) i vostri commenti: positivi o negativi che siano, saranno sicuramente costruttivi e ci aiuteranno a migliorare sempre di più la nostra rivista.

Buona lettura!
Maurizio Del Corso

DIRETTORE RESPONSABILE
Antonio Cirella

DIRETTORE TECNICO
Maurizio Del Corso

Segreteria di redazione
Giorgia Generali

Comitato scientifico
Simone Masoni (Microtest), Francesco Picchi (Microtest), Massimo Rovini (Università degli Studi di Pisa).

Art director
Patrizia Villa

Hanno collaborato in questo numero:
Silvano Breggion, Daniele Cappa, Marco Carminati, Roberto D'Amico (IW0GTF), Giovanni Di Maria, Antonio Giannico, Giuseppe La Rosa, Walter Lucetti, Massimiliano Miocchi, Giorgio Ober,

Direzione e redazione
INWARE Edizioni srl
Via Giotto, 7 - 20032 Cormano (MI)
Tel. 02.66504755
Fax 02.66508225
info@inwaredizioni.it
www.inwaredizioni.it
Redazione: fe@inwaredizioni.it

Pubblicità per l'Italia
Agostino Simone
Tel. +39 347 2230684
media@inwaredizioni.it

Europe and Americas
Elisabetta Rossi
Tel. +39 328 3245956
international@inwaredizioni.it

Asia
Cybermedia Communications Inc.
Tel. +886-(0)2-2691-2785
asia@inwaredizioni.it

Rest of the world
Inware Edizioni srl
+39 02 66504755
info@inwaredizioni.it

Stampa
Prontostampa
Via Redipuglia 150
24045 Fara Gera d'Adda (BG)

Distribuzione
Parrini & C s.p.a.
Via di Santa Cornelia, 9
00060 Formello (RM)

Ufficio abbonamenti
INWARE Edizioni srl
Via Giotto, 7 - 20032 Cormano (MI)
Per informazioni, sottoscrizione o rinnovo dell'abbonamento:
abbonamenti@inwaredizioni.it
Tel. 02.66504755
Fax. 02.66508225
L'ufficio abbonamenti è disponibile telefonicamente dal lunedì al venerdì dalle 14,30 alle 17,30.
Tel. 02.66504755
Fax 02.66508225
Abbonamento per l'Italia:
€ 55,00
Abbonamento per l'estero:
€ 115,00
Gli arretrati potranno essere richiesti, per iscritto, a **€ 9,00**
oltre le spese di spedizione

Autorizzazione alla pubblicazione
Tribunale di Milano n.647
del 17/11/2003

Mensile associato all'USPI
(Unione Stampa Periodica Italiana)

Privacy
Nel caso la rivista sia pervenuta in abbonamento o in omaggio, si rende noto che i dati in nostro possesso sono impiegati nel pieno rispetto del D.Lgs. 196/2003. I dati trasmessi a mezzo cartoline o questionari presenti nella rivista, potranno venire utilizzati per indagini di mercato, proposte commerciali, o l'inoltro di altri prodotti editoriali a scopo di saggio. L'interessato potrà avvalersi dei diritti previsti dalla succitata legge. In conformità a quanto disposto dal Codice di deontologia relativo al Trattamento di dati personali art. 2, comma 2, si comunica che presso la nostra sede di Cormano Via Giotto 7, esiste una banca dati di uso redazionale. Gli interessati potranno esercitare i diritti previsti dal D.Lgs. 196/2003 contattando il Responsabile del Trattamento Inware Edizioni Srl
(info@inwaredizioni.it).

Collaborare con FARE ELETTRONICA
Le richieste di collaborazione vanno indirizzate all'attenzione di Maurizio Del Corso (m.delcorso@inwaredizioni.it) e accompagnate, se possibile, da una breve descrizione delle vostre competenze tecniche e/o editoriali, oltre che da un elenco degli argomenti e/o progetti che desiderate proporre.

Elenco inserzionisti

Develer
Via Mugellese 1/A - 50013 Campi Bisenzio (FI)
Tel. 0553984627 – www.develer.it

Elettroshop
Via Giotto, 7 - 20032 Cormano (MI)
Tel. 02 66504755 - www.elettroshop.com

Eurogi
Via A. Casale 67 - 10010 Lessolo (TO)
Tel. 0125 58861 - www.eurogi.it

Grifo
Via dell'Artigiano 8/6
40016 San Giorgio Di Piano (BO)
Tel. 051-892052 - www.grifo.it

Innovability
Centro Direzionale Milanofiori Strada 1 Palazzo F2 - 20090 Assago (MI)
Tel. 02 48517925 - www.gowireless.it

Micromed
Via Valpadana 126B/2 - 00141 Roma (RM)
Tel. 06/90024006 - www.micromed.it

MikroElektronika
Visegradska, 1A - 11000 Belgrade
Tel. +381 11 3628830 - www.mikroe.com

Millennium Dataware
Corso Repubblica 48 - 15057 Tortona (AL)
Tel. 0131 860254 - www.mdsrl.it

PCB Pool
Bay 98-99 - Shannon Free Zone - Shannon County Clare
Tel. 02 64672645 – www.pcb-pool.com

Tecnoimprese
Via Console Flaminio, 19 - 20134 (MI)
Tel. 02 2101111 – www.fortronic.it

Teledyne LeCroy
via E. Mattei Valecenter 1/C - 30020 Marcon (VE)
Tel. 041 5997011 - www.lecroy.com

News

Il nuovo Camera Module Raspberry Pi è disponibile

RS Components ha ampliato la propria offerta con il nuovo modulo telecamera di Raspberry Pi, il cui prototipo era stato mostrato per la prima volta proprio presso stand RS a Electronica, la famosa fiera internazionale tenutasi lo scorso Novembre a Monaco di Baviera. Il modulo telecamera Raspberry Pi, acquistabile a soli $25, contiene un sensore della OmniVision, retroilluminato con architettura dei pixel a 1,4 micron, che assicura una risoluzione di 5 megapixel e riprese video HD. La telecamera si interfaccia con Raspberry Pi attraverso la connessione CSI e il protocollo I2C. La telecamera consente di registrare video in formato H264 a 720p/60 e 1080p/30.

http://it.rs-online.com

Il caricabatterie ecocompatibile di Fairchild Semiconductor riduce i tempi di ricarica e alimenta le periferiche USB

Il dispositivo di carica di tipo switching per batterie a ioni di litio a singola cella compatibile USB FAN54015 proposto da Fairchild Semiconductor fornisce ai progettisti un mix bilanciato di sicurezza, precisione, efficienza e dimensioni per ricaricare le batterie dei dispositivi mobili. Per ridurre i tempi di carica e la dissipazione di calore, il caricabatterie switching FAN54015 adotta un convertitore buck sincrono DC-DC ecocompatibile ad alta efficienza in grado di fornire 1,45A. Inoltre, per supportare le tecnologie delle batterie al litio emergenti così come quelle legacy, la tensione float regolata della batteria può essere programmata da 3,5V fino a 4,44V. L'innovativa architettura di sistema di questo dispositivo consente di invertire l'uso del convertitore buck in switch-mode per ottenere un regolatore boost USB On-the-Go (OTG) da 500mA adatto ad alimentare periferiche USB. Per proteggere la batteria e massimizzare i tempi di utilizzo del dispositivo mobile, l'unità FAN54015 supporta la carica in tre fasi: pre-condizionamento, corrente costante (CC) e tensione costante (CV). Un FET integrato per la protezione da sovratensioni blocca il flusso di corrente quando la tensione in ingresso raggiunge livelli pericolosi: questo elimina la necessità di un MOSFET di protezione esterno, riducendo così la complessità e i costi della componentistica. Il caricabatterie FAN54015 è completamente programmabile attraverso un'interfaccia I2C permettendo la configurazione di profili di carica custom.

www.fairchildsemi.com

News

Microchip amplia la famiglia di microcontroller PIC 8 bit con integrazione analogica intelligente

Microchip annuncia un ampliamento della sua famiglia di microcontroller avanzati Mid-Range core 8 bit PIC16F178X con aumentata densità di memoria Flash e periferiche intelligenti analogiche e digitali, come ADC 12 bit on-chip, PWM 16 bit, DAC 8 bit e 5 bit, amplificatori operazionali, e comparatori ad alta velocità con tempo di risposta di 50 ns; oltre ad interfacce per periferiche EUSART (incluso LIN), I2C e SPI. I PIC16F178X sono i primi MCU PIC ad implementare il nuovo Programmable Switch Mode Controller, che è un PWM 16 bit avanzato con funzionamento a 64 MHz e capace di elevate prestazioni.
Questa combinazione di caratteristiche e funzionalità permette più elevate efficienza e prestazioni, abbinati ad una riduzione di dimensioni e costi. I nuovi MCU usano anche la tecnologia eXtreme Low Power per correnti attive e in sleep di soli 32 µA/MHz e 50 nA, rispettivamente, contribuendo al prolungamento della vita delle batterie e a ridurre il consumo di corrente in standby. Disponibile in package di 28- e 40-pin, l'integrazione analogica intelligente degli MCU abbinata a periferiche Core Independent, che comprendono PSMC, DAC, Op Amp, comparatori ad alta velocità e ADC 12 bit, consente loop di controllo intelligenti autosostenuti con un intervento minimo di CPU. Questo consente un ottimale controllo della applicazione liberando la CPU perché possa incrementare il valore dell'applicazione, come il controllo dello stato di salute di sistemi, comunicazioni, o controllo di interfacce umane. Inoltre, gli MCU dispongono di un oscillatore interno a 32 MHz, Flash da 2 – 16K Word (3.5 – 28K Byte), 256 – 2K Byte di RAM e 256 Byte di EEPROM dati.

www.microchip.com/get/T2VS

Harwin e Mouser co-sponsor del corsa automobilistica IndyCar: e anche voi potrete assistere alla gara!

Harwin comunica con soddisfazione di co-sponsorizzare, con Mouser Electronics, Tony Kanaan, il pilota leader della corsa automobilistica IndyCar, per l'edizione del 2013 dell'Izod IndyCar Series.

Il pilota campione Tony Kanaan, che ha partecipato all'edizione n. 11 della corsa IndyCar Mouser Electronics/Geico/KV Racing Technology Chevrolet/Firestone, ha ottenuto sette volte la prima posizione nei sesti di finale nella stagione 2012, incluse tre presenze sul podio, un secondo posto assoluto a Milwaukee, due volte il notevole terzo posto a Iowa e il podio assoluto nella gara Indianapolis 500.
I prodotti per le connessioni di Harwin, come il Datamate e il Gecko, sono utilizzati in numerose applicazioni per auto sportive, compresi la Formula 1 e l'IndyCar, per assicurare l'integrità di segnale anche in condizioni di stress e di sforzo delle gare di alta velocità, quando le auto sono soggette a numerosi G nel percorrere le curve.
Harwin offre anche ai visitatori del suo sito web l'opportunità di presenziare alla serie Izod IndyCar. Semplicemente, registrandosi su:

www.harwin.com/aboutus/indy

GMM AM644: Mini Modulo Atmel ATmega644P

Potente ed economico Mini Modulo basato sul controllore Atmel Atmega644P. In un contenitore DIP da 40 piedini, e con 64K di FLASH, un completo SBC programmabile In Circuit con linguaggi evoluti come C, BASIC, ecc. Il GMM AM644 è in grado di essere utilizzato come Macro Componente, direttamente sulla scheda dell'utente.
E' alimentato a 5 Vdc e ha tutto ciò che serve per funzionare e per comunicare tramite una linea seriale a livello TTL oppure in RS232 II

News

grossissimo vantaggio di questo Mini Modulo, è quello di possedere un programma di Boot-Loader che gli consente di essere Programmato/Cancellato usando la sola linea di comunicazione seriale. Questo significa che qualsiasi sperimentatore può farsi un circuito, perfettamente funzionante, e programmarlo usando la sola linea seriale. E' il componente ideale per risolvere i problemi di automazione sia industriale che domestica. A questo scopo è sufficiente provvedere, tramite una circuiteria esterna, a bufferizzare le linee di I/O disponibili. Se non si vuole costruire dell'hardware, è possibile utilizzare il GMB HR128 il quale provvede ad alimentare e bufferizzarizzare le linee di I/O del GMM AM644 con 16 ingressi Optoisolati, e visualizzati tramite LED, indifferentemente usati come ingressi NPN o PNP, 8 relé da 5 A, 2 linee seriali TTL o RS232, una linea in I2C BUS, RTC con batteria tampone: area interna di 8K di FRAM in I2C BUS. Se, invece, servono più risorse analogiche si può ricorrere alla GAB H844 che con le sue 8 linee di A/D converter, 4 Opto-In e 4 Relé Output diventa una risorsa ideale. Molto interessanti anche le dotazione di software di programmazione, e di esempi, che comprendono vari Compilatori C, BASIC tra cui il BASCOM AVR con disponibile anche il Corso Gratuito.
Il GMM AM644 è distribuito da Grifo - Via dell'Artigiano 8/6 - 40016 San Giorgio di Piano (BO)

www.grifo.it

Il nuovo sistema di contatto senza connettore di AVX ottimizza l'allineamento laterale del PCB nelle applicazioni di illuminazione a LED lineare

AVX Corporation ha sviluppato un sistema di contatto senza connettore complanare che ottimizza l'allineamento lineare del PCB e massimizza le tolleranze di accoppiamento nelle applicazioni di illuminazione a LED lineare. Confezionato su nastro e bobina, per la collocazione singola automatica, il nuovo sistema di contatto orizzontale 70-9159 presenta un rating di corrente di 5 A, assorbe importanti tol-

News

leranze negli assi X e Y in assemblaggio, fornisce un'affidabile interfaccia di contatto attivo da oro a oro per una massima integrità di contatto negli ambienti disagevoli d'illuminazione, e supporta le applicazioni sia da scheda a scheda che da filo a scheda.
I nuovi contatti 70-9159 di AVX eliminano l'isolatore dalla soluzione di connessione, permettendo la collocazione dei LED al centro del PCB e la collocazione singola dei contatti sul bordo esterno, massimizzando in modo efficiente il funzionamento e attenuando i costi.
Presentando un'altezza del contatto di 1,2 mm oltre il PCB per evitare ombre, i contatti possono essere collocati singolarmente al fine di supportare qualsiasi rating di tensione con un rating di corrente di 5 A.
In grado di supportare temperature di funzionamento in una gamma da -40 °C a +125 °C, i contatti del 70-9159 in lega di rame hanno i contatti del socket dorati e 1 mm di tolleranza da lato a lato, spine in piombo con doratura dell'area di accoppiamento, 1 mm di tolleranza dell'accoppiamento e durata di cinque cicli.

www.avx.com

La tecnologia "Projected Capacitive Touch Screen" di Molex rappresenta una soluzione personalizzabile in grado di garantire un elevato livello di precisione

Molex Incorporated ha presentato i suoi Schermi capacitivi a sfioramento che sfruttano la tecnologia di commutazione capacitiva di Molex attualmente esistente per ottenere una funzionalità multi-touch reattiva e intuitiva da utilizzare.
Gli schermi "touchscreen" consentono ai produttori (OEM) di soddisfare le richieste specifiche dei clienti offrendo loro un software integrato personalizzato, diverse tipologie di schermi, un'ampia gamma di finiture e diverse opzioni per l'interfaccia di uscita.
Gli schermi capacitivi a sfioramento di Molex presentano uno strato conduttivo inciso che consente il rilevamento del tocco attraverso gli strati protettivi, garantendo in tal modo una lunga durata, fino a 200 milioni di attuazioni.
La capacità multi-touch a 10 punti consente funzioni di gestione avanzate come pan, rotazione, espansione e flick. Il prodotto è disponibile anche in un'ampia serie di opzioni personalizzabili in grado di soddisfare specifici criteri di progetto e di budget, tra cui: dimensioni dello schermo comprese tra 2,00 e 32,00", con risoluzione di 4096 per 4096 dpi, vasta gamma di materiali dello schermo tra cui vetro-vetro, vetro-pellicola e pellicola-pellicola, vari tipi di vetro tra cui, per uso generale, chimicamente rinforzato e temprato, oltre a vari trattamenti superficiali finali come antiriflesso, antiabbagliamento e trasparente, software integrato configurabile per soddisfare specifici requisiti di uscita elettrica, varie interfacce di uscita tra cui USB, I2C e segnali discreti.

www.molex.com

TI presenta due driver LED altamente integrati per lampade/luci da incasso a LED retrofit

I nuovi driver LED semplificano il progetto di lampade a LED retrofit E14, GU10, A19, PAR20/30/38, MR16 e AR111, riducendo il numero di componenti e le dimensioni della soluzione e migliorando allo stesso tempo la compatibilità con i controlli di illuminazione tradizionali.
Il TPS92075 è il più piccolo controller al mondo a corrente costante per illuminazione a LED, con commutazione di fase e modalità buck o buck-boost offline.
Tra le caratteristiche: package TSOT a 6 pin, funzionamento con correzione del fattore di potenza a 120 VCA o 230 VCA, compatibilità con dimmer TRIAC e trailing-edge. Il TPS92560 è il driver LED di facile utilizzo ottimizza la compatibilità con i trasformatori elettronici tradizionali.
Tra le caratteristiche: ssemplice schema di controllo a isteresi della corrente di ingresso, raddrizzatori di ingresso low-side attivi integrati, funzionamento in modalità buck o buck/boost.

www.ti.com

Eventi

ZERO EMISSION ROME 2013

Giunto alla sua sesta edizione, ZEROEMISSION ROME è l'evento di riferimento per tutte le aziende e gli operatori interessati allo sviluppo delle energie rinnovabili, all'emission trading e alla sostenibilità ambientale in Italia e nel grande e promettente mercato del bacino del Mediterraneo. ZEROEMISSION ROME 2013 è l'insieme di eventi specializzati dedicati all'energia eolica, all'energia fotovoltaica, al solare termodinamico, all'emission trading, cambiamenti climatici e CCS, agroenergie e biocarburanti. Insieme occuperanno ben quattro grandi padiglioni di Fiera di Roma su un'area di oltre 40.000 metri quadri.

Dove: Roma • Quando: 9-11 Ottobre 2013 • Orari: dalle 9.30 alle 18.30 • Organizzazione: ZeroEmission • info: www.zeroemissionrome.eu

AUTOMAZIONE IN FIERA

SAVE
Mostra Convegno delle Soluzioni e Applicazioni Verticali di Automazione, Strumentazione, Sensori.

SAVE è un appuntamento innovativo che unisce una parte espositiva in fiera ad una forte componente formativa. Area espositiva dove incontrare agli stand i principali leader di settore, centinaia di convegni e workshop accessibili per gli operatori qualificati, cinque eventi internazionali e un evento speciale in contemporanea, gli operatori professionali accedono gratuitamente alla mostra e usufruiscono di tutti i servizi. L'esigenza percepita oggi è rendere adeguate le modalità fieristiche, renderle attuali e sempre più proficue per creare una fiera sull'automazione e sulla strumentazione utile agli operatori.

Dove: Verona • Quando: 29-30 ottobre 2013 • Orari: dalle 9.00 alle 18.00 • Organizzazione: EIOM • info: www.exposave.com

EXPOELETTRONICA - BOLZANO

Il circuito di fiere Expo Elettronica, a giugno 2013, cresce nuovamente con una nuova tappa prevista presso l'Ente Fiera di Bolzano. Expo Elettronica ha un pubblico vasto ed eterogeneo: appassionati del "fai da te", elettro-riparatori, "smanettoni", radioamatori, "cacciatori" di buone occasioni o pezzi rari; questo perché propone un panorama merceologico e un calendario di eventi collaterali veramente ricchissimo.

Dove: Bolzano • Quando: 8-9 Giugno 2013 • Orari: dalle 9.00 alle 18.00 • Organizzazione: Blunautilus • info:www.expoelettronica.it

Eventi

ELETTRO-BIT EXPO

Anche quest'anno si rinnova l'appuntamento con la Fiera dell'elettronica di Reggio Emilia. La manifestazione, all'interno del complesso fieristico Fiere di Reggio Emilia (Reggio Emilia), si svolgerà con orario continuato, dalle 9,00 alle 18,30. Saranno presenti oltre 100 espositori, su un'area di 7.000 metri quadrati e con oltre 800 metri lineari di banchi espositivi.

Dove: Reggio Emilia • Quando: 8-9 Giugno 2013 • Orari: dalle 9.00 alle 18.30 • Organizzazione: Openoffice • info: www.fieraelettronicareggioemilia.it

HAM RADIO

Un appuntamento storico per i radioamatori. L'Ham Radio di Friedrichshafen è la fiera europea di riferimento per radioamatori ed hobbisti elettronici che attrae visitatori da tutto il mondo. In occasione della fiera si terranno workshop e conferenze di sicuro interesse per gli appassionati del settore.

Dove: Friedrichshafen (Germania) • Quando: 28-30 Giugno 2013 • Orari: dalle 9.00 alle 18.00 • Organizzazione: Messe Friedrichshafen • info: www.hamradio-friedrichshafen.de

MOSTRA NAZIONALE MERCATO RADIANTISTICO

RADIANTISTICA EXPÒ
MOSTRA MERCATO RADIANTISTICO

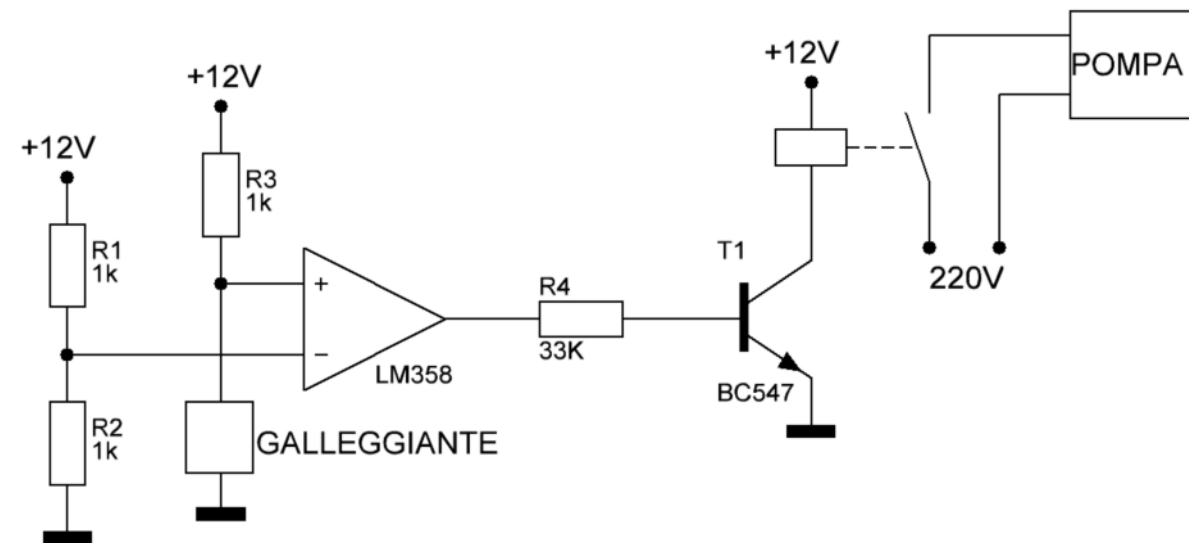
Mostra mercato radiantistico dell'elettronica, CD, editoria specializzata, telefonia cellulare.

Dove: Montichiari (BS) • Quando: 31 Agosto – 1 settembre 2013 • Orari: dalle 9.00 alle 18.00 • Organizzazione: Centrofiera • info: www.centrofiera.it

MOSTRA DELL'ELETTRONICA

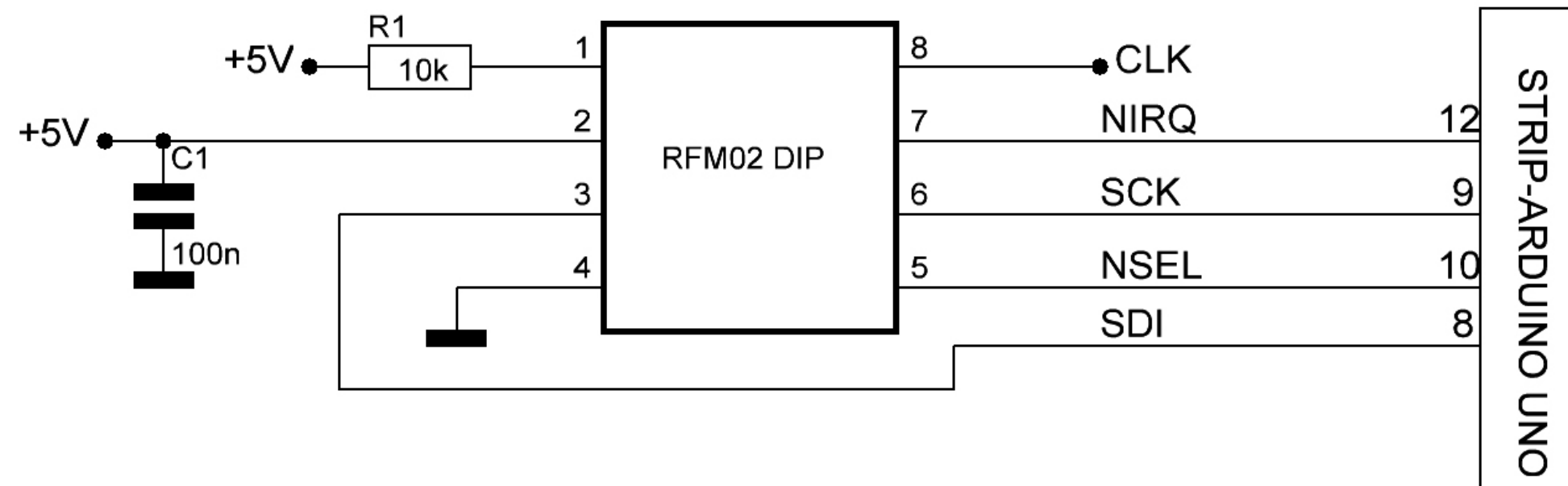
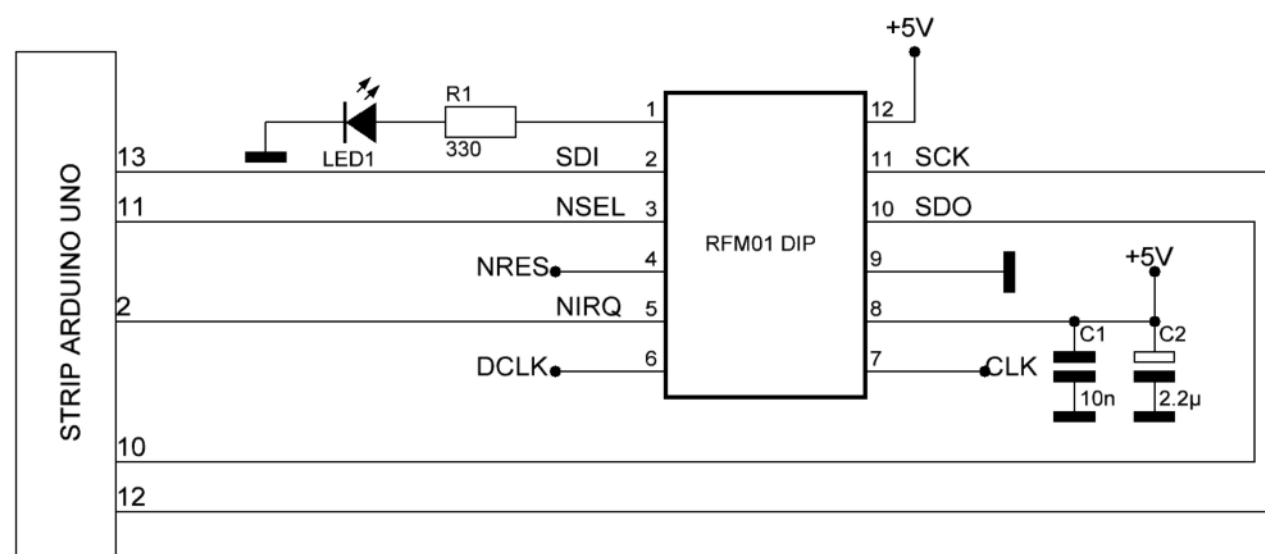
Computer e telefonia, informatica, elettronica, antenne e TV/SAT, Hi-Fi, sicurezza informatica, editoria specializzata. Queste le tematiche della terza edizione della Mostra dell'elettronica in Valle d'Itria.

Dove: Martina Franca (TA) • Quando: 21-23 Giugno 2013 • Orari: dalle 9.00 alle 18.00 • Organizzazione: Anse Fiere • info: www.ansefiere.it

Idee di progetto

ATTIVATORE AUTOMATICO PER POMPA

Lo schema in figura è un semplice circuito che permette di attivare una pompa idrica quando il liquido nel serbatoio scende al di sotto di un certo livello. Questo circuito si compone di un amplificatore operazionale LM358, un partitore resistivo, un galleggiante e un circuito di interfacciamento realizzato con un transistor e un relè. Il circuito può essere alimentato con una tensione di 12V.

TRASMETTITORE FSK

In figura è riportato lo schema per la connessione tipica di un modulo radio FSK "RFM02" con ARDUINO. Questo modulo radio funziona con le bande ISM ,ovvero bande assegnate per scopi non commerciali ma per applicazioni scientifiche. Sebbene tali moduli presentino il vantaggio di avere dimensioni ridotte e la possibilità di interfacciarsi con qualsiasi microcontrollore, lo svantaggio è quello di essere molto sensibili ai disturbi, specialmente se le dimensioni delle antenne non sono simmetriche.

Idee di progetto

RICEVITORE FSK

Nella figura è riportata la connessione del modulo radio ricevitore "RFM01" con ARDUINO. Questo modulo presenta le stesse caratteristiche del suo trasmettitore. La tensione di alimentazione massima di entrambe i moduli radio è di 5V.

CONVERTITORE 18BIT

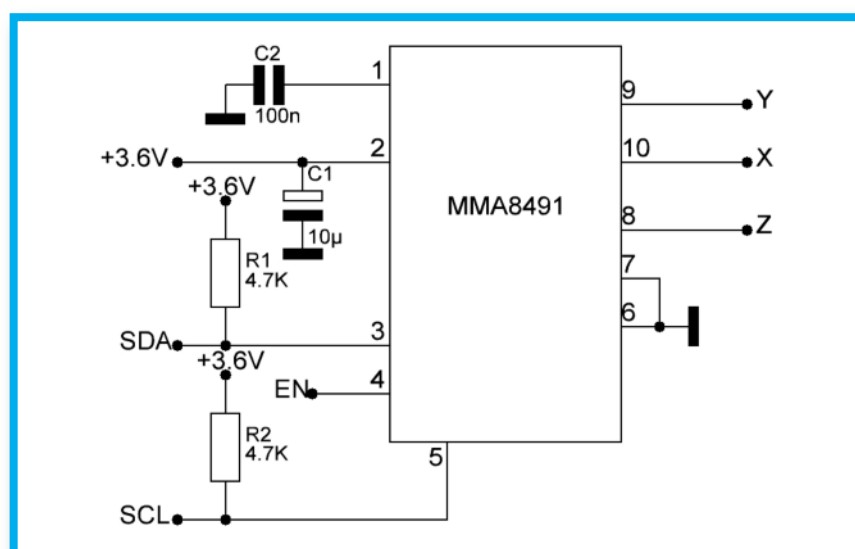
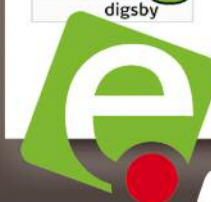
Il circuito rappresentato in figura è un convertitore A/D a 18 bit realizzato con MAX11156. Tale integrato è stato sviluppato proprio per far fronte alle nuove tecnologie che richiedono una velocità di elaborazione molto elevata.

Idee di progetto

AMPLIFICATORE DUALE DA 3W

Nella figura seguente viene riportato un semplice amplificatore da 3W. Tele circuito può essere utilizzato come amplificatore portatile per dispositivi come: lettore MP3, IPOD, ecc.

ACCELEROMETRO 3 ASSI

Il circuito in figura è stato realizzato utilizzando l'applicazione tipica di un accelerometro. Questo semplice circuito richiede pochissimi componenti per funzionare , con una tensione di alimentazione di +3.6V,questa caratteristica gli permette di essere utilizzato in molteplici dispositivi come: videogiochi, portatili, tablet, ecc.

di NICOLA TARASCHI

MONITORAGGIO DI ARDUINO

Come controllare via PC lo stato di una applicazione basata su Arduino

Arduino è un microcontrollore ampiamente diffuso sia per la economicità del prodotto, per la sua affidabilità hardware e software, oltre che per la reperibilità di numerose schede di supporto che permettono il controllo di una vasta gamma di sensori e di azionamenti, e la disponibilità del software di supporto.
Come abbiamo sottolineato in un precedente articolo sul CUBLOC, nell'ambito dei microcontrollori l'implementazione di applicazioni non prevede, di base, una interfaccia utente .Lo sviluppo del software può avvalersi di un programma di monitor che può visualizzare sullo schermo del PC l'evoluzione delle variabili del processo e dei segnali analogici e digitali di ingresso e uscita . Lo stato del processo controllato dal software può essere controllato, in assenza di un colloquio con un PC, solo da indicatori Led o display LCD o, come estrema possibilità, dalla correttezza delle azioni del processo stesso.
Nell'ambito di applicazioni didattiche o nello sviluppo di applicazioni industriali interattive ha grande importanza la visualizzazione dello stato del processo e l'interazione con l'utente. Non mancano soluzioni in cui lo stato dell'hardware può essere visualizzato e controllato dall'utente come ad esempio LABVIEW, che è un esempio di software interattivo che mette a disposizione dello sviluppatore un ambiente, anche abbastanza complesso, finalizzato al controllo dell'hardware.
PROCESSING è un linguaggio che permette l'interazione e il controllo di ARDUINO tramite un linguaggio di programmazione del tutto simile a quello di ARDUINO.
La nostra soluzione, che presentiamo in questo articolo, fa uso di un linguaggio di programmazione classico, DELPHI, e di EXCEL. Mentre DELPHI ed analogamente VISUAL BASIC e simili rappresentano linguaggi di programmazione classici in cui il programmatore deve costruirsi l'interfaccia utente, EXCEL, abbinato al residente VBA, linguaggio del tutto simile al VISUAL BASIC, ha già una potente interfaccia utente pronta all'uso.
Attraverso DELPHI o EXCEL, o simili, il PC dialoga con il software residente su ARDUINO, o sarebbe meglio definirlo un firmware, attraverso la porta seriale.
Il firmware, a richiesta del PC, invia i valori letti dalle porte oppure ne esegue la scrittura . Il programma residente sul PC gestisce in modo integrale tutto il processo di controllo ed acquisizione, limitando il colloquio con ARDUINO alle operazioni di lettura e /o scrittura delle porte. Questa soluzione non comporterebbe nessuna conoscenza del linguaggio di ARDUINO, ma semplicemente una scheda ARDUINO in cui sia residente solo questo software. Il software può essere arricchito per interfacciare specifici sensori :ad esempio è disponibile per ARDUINO il software per il sensore di temperatura digitale 18B20. In questo caso il firmware accetterebbe la richiesta di lettura, la elabora, e risponde con il valore letto.
Il colloquio PC- ARDUINO diventa un colloquio master –slave, in cui ARDUINO si occupa di funzionalità di basso livello come la gestione dell'hardware, mentre il PC svolge il ruolo di assoluto gestore del processo. Il PC, disponendo di una interfaccia utente

con tutte le risorse di WINDOWS, consente l'accesso alla gestione di dati, la loro rappresentazione grafica, la memorizzazione e così via. In sede di sviluppo del software sul PC il programmatore ha la disponibilità delle risorse tipiche di linguaggi di alto livello come DELPHI o VISUAL BASIC che permettono il debugging del programma, l'esecuzione passo-passo, la visualizzazione delle variabili, l'inserimento di punti d'interruzione dell'esecuzione e così via .Il debugging del programma su ARDUINO è semplice solo per piccoli programmi mentre risulta già arduo per programmi che sul PC sarebbero minimi.
Qualora il processo non preveda, di norma, l'interfacciamento con il PC, l'implementazione di tale firmware potrebbe essere utile in fase di diagnosi del processo stesso. E' questo il caso, ad esempio, delle apparecchiature che interrogano, quando collegate, lo stato dei segnali provenienti da apparecchiature dotate di porta RS232.
Bisogna comunque considerare che anche a livello di immagine lo stesso processo basato sulla interfaccia con il PC acquista uno spessore diverso rispetto alla stessa soluzione sul solo ARDUINO.

IL SOFTWARE SU ARDUINO

Nel listato 1 vengono riportate le due routine principali del programma. La routine SETUP, eseguita all'avvio del programma inizializza la porta seriale. La routine LOOP, che viene eseguita ciclicamente dal microcontrollore aspetta i bytes inviati dal PC e li memorizza nel vettore RICEV, che è un array di bytes. Il contatore COUNT1 incrementa la posizione di memorizzazione del byte in arrivo nel vettore RICEV . Quando sono stati ricevuti 7 byte il vettore RICEV viene analizzato dalla routine ESAMINA e contemporaneamente resettato il contatore COUNT1.
La routine ESAMINA esegue il comando associato alla stringa e risponde, se vi è una richiesta di lettura dati, con analoga stringa di 7 caratteri che contiene i dati richiesti. Il primo carattere della stringa è significativo del comando associato . I comandi sono:

- R=lettura delle porte digitali
- A=lettura delle porte analogiche
- D=scrittura sulle porte digitali
- B=scrittura sulle porte analogiche

figura 1: il pinout di arduino con le porte

La lettura di una porta analogica

La stringa inviata ad ARDUINO è del tipo A0Xnnnn . Il primo carattere della stringa è uguale ad A (codice ascii 65).
Il secondo carattere è lasciato libero .Il terzo carattere è il numero della porta :A= porta 0 B=porta 1 e così via. Gli altri caratteri non hanno significato. Decodificata la porta da leggere ne avviene la lettura e la variabile SENSORVALUE riporta il valore, fra 0 e 1023 (il convertitore è a 10 bit) .A questo valore viene sommato 1023 in modo che il risultato sia sempre a 4 cifre. La stringa di uscita sarà formata dai primi 3 caratteri uguali a quelli della stringa di ingresso, mentre gli ultimi 4 riportano il valore del dato analogico.

La lettura di dati digitali

La stringa ricevuta è del tipo R0Xnnnn, solo che in questo caso la porta deve essere posta in INPUT con il comando PINMODE.
La stringa restituita è del tipo R00LLLL (porta OFF) oppure R00HHHH (porta on).

LISTATO 1
(la parte principale del programma su ARDUINO)

```
void setup()
{
Serial.begin(9600);
}
void loop()
{
if (Serial.available()>0)
{
inbyte =Serial.read();
count=count+1;
ricev[count]=inbyte;
//Serial.write(inbyte);
}
if (count>6) {
 esamina();
 count=0;
 }
```

Scrittura di dati digitali

La stringa ricevuta è del tipo D0XHHHH (porta ON) oppure D0XLLLL (porta OFF). La porta deve essere posta in OUTPUT con il comando PINMODE.

Scrittura porta analogica

La stringa inviata ad ARDUINO è del tipo B0XnDDD . Il primo carattere della stringa è uguale a B (codice ascii 66) e le stesse considerazioni fatte sopra valgono per i caratteri dal 2° al 3°. Il 4° carattere, in questo caso è lasciato vuoto. Nel caso di ARDUINO il dato analogico in scrittura è compre-

Figura 2:la scheda di prova

Figura 3:l'interfaccia utente sul PC di controllo della scheda

so fra 0 e 255 e quindi bastano i caratteri dal 5° al 7° per rappresentare il dato. La routine NUM trasforma i bytes nel numero che viene inviato sulla porta di uscita.

LA SCHEDA DI PROVA

Arduino (figura 1) uno ha 6 porte analogiche a 10 bit e 14 porte digitali di cui 3 pwm a 8 bit. Una porta USB permette il download del software e la comunicazione con il PC .La versione maggiore, ARDUINO MEGA, ha lo stesso software ma un numero di porte più elevato. Una prima prova del software è stata fatta con una scheda(figura 2) in cui sono riportati 2 ingressi analogici con sensori di temperatura NTC 10K, 3 ingressi analogici simulati con trimmer, 4 ingressi digitali con pulsanti e 5 uscite digitali +1 analogica realizzate con led. L'interfaccia utente per il controllo della scheda è riportata nella figura 3. L'interfaccia software della scheda è riportata nella figura 3. Il software è ugualmente valido anche se anziché ARDUINO è collegato un CUBLOC, o altro microcontrollore, con analogo firmware. Selezionando la porta sulla sinistra(2, 3, 4, 5, 8, 11) e selezionando lo stato della relativa porta (ON oppure OFF) si attiva il relativo comando. Contemporaneamente viene letto lo stato delle porte di ingresso (6, 7, 9, 10) e, se la porta analogica specificata è diversa da zero, questa viene settata al valore imposto con la barra di scorrimento (100% corrisponde al valore 255).Con il pulsante TEST vengono invece lette le porte analogiche il cui valore viene riportato nello spazio libero di destra. L'applicazione DELPHI si basa su routines che svolgono il ruolo di comunicazione PC-ARDUINO, abbastanza semplici nel loro significato. Prima di iniziare le operazioni di lettura/scrittura occorre chiamare la routine di inizializzazione della porta seriale, che viene selezionata come nella videata.

La lettura, ad esempio, del valore della porta analogica 2, avviene 0

```
Ok:=inizia_seriale('COM2');
a:=input_analogico(2)
fine_seriale
```

UNA PRIMA APPLICAZIONE

Un trasduttore analogico è un componente che, produce una tensione analogica in funzione della variazione di una grandezza fisica. Ad esempio un trasduttore di posizione potenziometrico produce in uscita una tensione in funzione dello spostamento del cursore mobile e quindi permette, nota la tensione, di ricavare lo spostamento

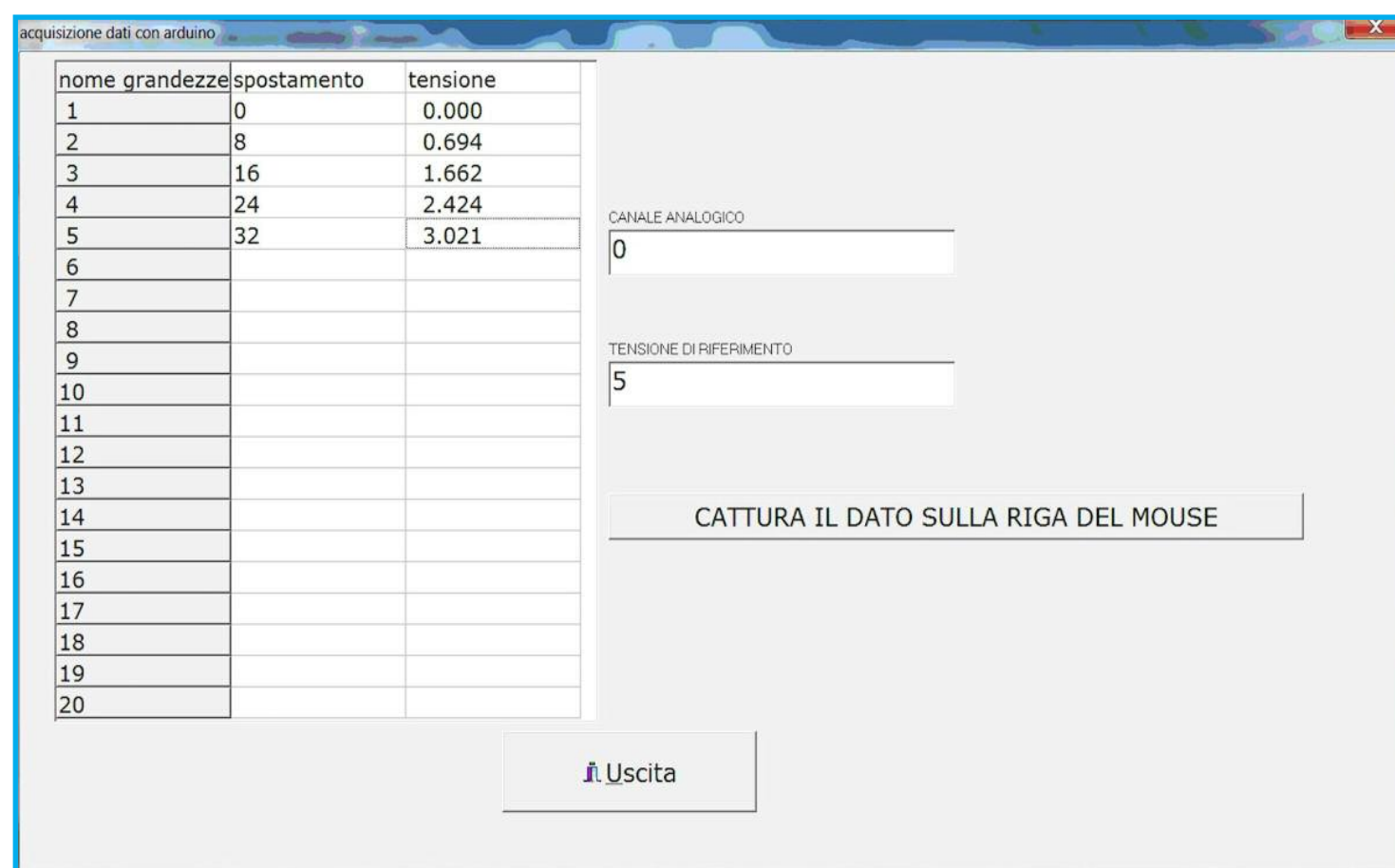



Figura 4:l'interfaccia utente di acquisizione valore analogico

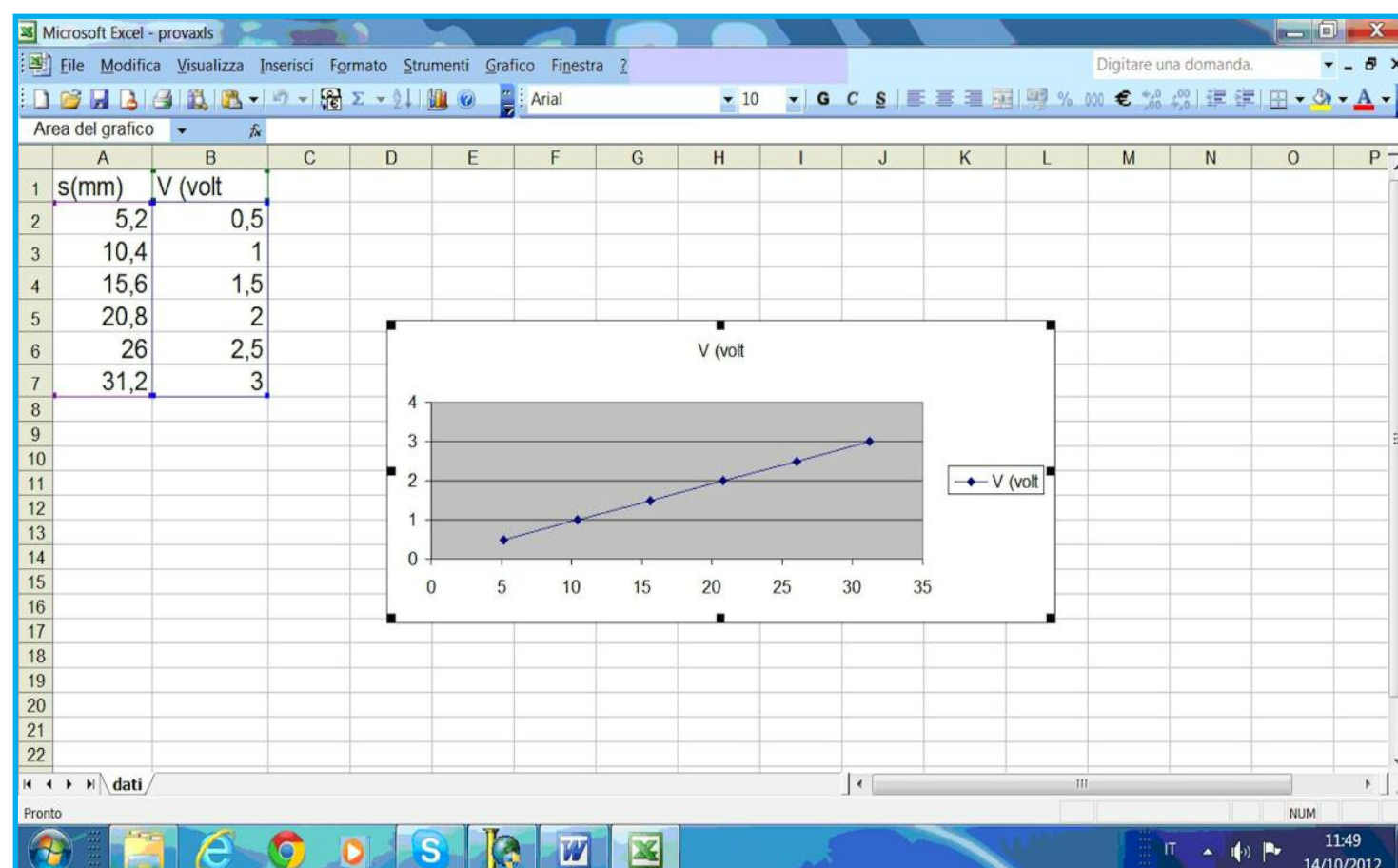



Figura 5:i dati sul foglio EXCEL

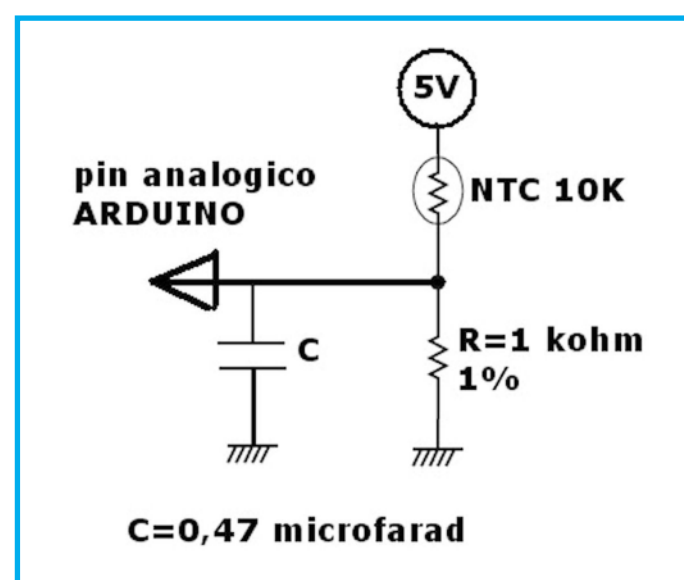



Figura 7:il collegamento del NTC

Questa prima applicazione utilizza ARDUINO come sistema di acquisizione dati da una porta analogica .I dati catturati vengono poi inviati ad un file EXCEL che ne permette la gestione più opportuna.

Nella colonna dello SPOSTAMENTO (figura 4) vengono collocati i dati della grandezza fisica tipica del trasduttore(in questo caso uno spostamento) .

Cliccando sulla riga corrispondente e sul bottone: CATTURA IL DATO… viene letta la tensione analogica dal canale specificato e il valore posto nella relativa cella. La figura 5 riporta i dati trasferiti su EXCEL e la loro elaborazione per ottenere la loro rappresentazione grafica.

Acquisizione dati –temporale

In questa maschera (figura 6) vengono acquisiti sia i valori del tempo, secondo l'intervallo temporale prefissato, che quella della corrispondente tensione analogica, proveniente, ad esempio, da un trasduttore analogico.

In tal caso ARDUINO svolge la funzione di DATA-LOGGING, che può essere ulterior-

mente migliorata trasformando la tensione nella grandezza fisica corrispondente.

Controllo ON-OFF

L'ultima applicazione è una applicazione di controllo ON-OFF di temperatura, secondo il setpoint e il differenziale impostato. Il sensore di temperatura è un NTC e un partitore di tensione, secondo lo schema riportato nella figura 7 . L'uscita di ARDUINO comanda un generico dispositivo che può essere ad esempio un RELAY.
La parte del listato relativa è facilmente comprensibile. L'interfaccia utente è riportata nella figura 8.

Il software su EXCEL

EXCEL è un programma che non ha bisogno di essere presentato .Il suo uso è generalmente diretto ad applicazione di calcolo e gestione dei dati, nondimeno in questo caso diventa uno strumento di acquisizione dati . All'interno di EXCEL, come di ogni altro programma del pacchetto OFFICE, vi è un linguaggio denominato VBA(VISUAL BASIC FOR APPLICATION) del tutto identico al VISUAL BASIC con cui è possibile controllare le applicazioni .
Il software su EXCEL si appoggia a routine di comunicazioni, scritte in VBA, che sono formalmente identiche a quelle già illustrate in DELPHI. Un considerevole vantaggio è di poter disporre di tutte le potenzialità del foglio elettronico per la gestione dei dati e la presentazione dell'interfaccia utente .
L'interfaccia utente si presenta nella figura 9. In questo caso l'utente, e non il programmatore, ad esempio, può variare le porte .

acquisizione dati con arduino

2	4.071
4	4.076
6	4.076
8	4.076
10	4.076
12	4.076
14	4.076
16	4.076
18	4.076
20	4.990
22	2.957
24	2.204
26	1.544

CANALE ANALOGICO: 0
TENSIONE DI RIFERIMENTO: 5
tempo di campionamento(sec): 2
Start
Uscita

Figura 6:l'interfaccia utente di acquisizione dati in modalità DATALOGGING

LISTATO 2
(lettura dato analogico)

```
//lettura analogica
if (inbyte==65||inbyte==193) {Serial.print("A00");
inbyte=ricev[3];
if (inbyte>128) {inbyte=inbyte-128;};
porta=inbyte-65;
sensorvalue=analogRead(porta)+1023;
Serial.print(sensorvalue);
 }
```

LISTATO 3
(scrittura analogica)

```
void esamina_numero()
{
if (ricev[j]>175) {ricev[j]=ricev[j]-176 ; }
if (ricev[j]<60&&ricev[j]>47) {ricev[j]=ricev[j]-48; };
}
void numero()
{
for (j=5;j<8;j++) {
esamina_numero();}
num6=100*ricev[5]+10*ricev[6]+ricev[7];
}
//scrittura ANALOGICA
if (inbyte==66||inbyte==194) {
inbyte=ricev[3];
if (inbyte>128) {inbyte=inbyte-128;};
porta=inbyte-65;
numero();
pinMode(porta, OUTPUT);
analogWrite(porta, num6);
num6=num6+100;
 }
```

Figura 8:l'interfaccia utente sul PC del controllo ON-OFF

	A	B	C	D	E
2	porta seriale	3			
3		porta	valore	tensione	
4	porta analogica	0	316	1,544	volt
5		1	858	4,194	
6		2	849	4,150	
7		4	0	0,000	
8		5	85	0,415	
9	porta digitale ingresso	6	OFF		
10		7	OFF		
11		9	ON		
12		10	ON		
13	porta digitale uscita	11	1		
14		8	1		
15		3	1		
16		0	0		
17		0	0		
18		0	0		
19	porta PWM	8	99	%	

Figura 9: il foglio EXCEL

Schiacciando RUN il programma legge le porte e le setta secondo un timer programmato con intervallo di 1 secondo. Schiacciando CLOSE il timer si arresta e così la scansione dei valori.

Poiché il programma contiene il software in VBA il livello di protezione MACRO deve essere settato su basso, altrimenti il software non potrà funzionare. Un esempio della routine è riportata nel listato 8.

LISTATO 4
(lettura digitale)

```
//lettura digitale
if (inbyte==82||inbyte==210) {
inbyte=ricev[3];
if (inbyte>128){inbyte=inbyte-128;}
porta=inbyte-65;
pinMode(porta, INPUT);
sensorvalue=digitalRead(porta);
if (sensorvalue==HIGH) {Serial.print(“R00HHHH”);}
if (sensorvalue==LOW) {Serial.print(“R00LLLL”);}
}
```

LISTATO 5
(scrittura digitale)

```
//scrittura digitale
if (inbyte==68||inbyte==196) {
inbyte=ricev[3];
if (inbyte>128){inbyte=inbyte-128;}
porta=inbyte-65;
inbyte=ricev[4];
if (inbyte==72||inbyte==200) {uscita=1;}
if (inbyte==76||inbyte==204) {uscita=0;}
pinMode(porta, OUTPUT);
if (uscita==0) {digitalWrite(porta, LOW);}
if (uscita==1) {digitalWrite(porta, HIGH);
}
```

LISTATO 6
(le routines di comunicazione)

```
FUNCTION INIZIA_SERIALE(PORTA :STRINGA):BOOLEAN;
FUNCTION INPUT_ANALOGICO(CANALE:BYTE):INTEGER;
FUNCTION INPUT_TENSIONE(CANALE:BYTE;VRIF:REAL):REAL;
FUNCTION INPUT_DIGITALE(PORTA:BYTE):INTEGER;
PROCEDURE SCRITTURA_DIGITALE(P:WORD;MODO:WORD);
PROCEDURE SCRITTURA_ANALOGICA(Porta, MODO:WORD);
PROCEDURE FINE_SERIALE;
PROCEDURE salvacomefileEXCELFILE(NOME:STRING);
```

LISTATO 7
(il controllo ON-OFF in DELPHI sul PC)

```
s:=temp_ntc10k(canale_in);
val(s, temperatura, cod);
labelededit1.Text:=s;
if temperatura<setpoint-differenziale/2 then
scrittura_digitale(porta_out, 1);
if temperatura>setpoint+differenziale/2 then
scrittura_digitale(porta_out, 0)
```

LISTATO 8
(la lettura in VBA dei valori analogici)

```
Private Sub scansiona()
….
For i = 1 To 5
n = Cells(3 + i, 2).Value
a = lettura_analogica(n)
Cells(3 + i, 3).Value = a
volt = 5 * a / 1023
Cells(3 + i, 4).Value = volt
Next i
….
End Sub
```

di ALBERTO TRASIMENI

MICROCONTROLLORI

PORT EXPANDER: GESTIRLO CON UN PIC

In progetti in cui viene impiegato un microcontrollore con un numero limitato di I/O, può essere necessario utilizzare un port expander per incrementare il numero di ingressi/uscite disponibili. Ecco come fare

L'integrato MCP23S17, è un "Port Expander" bidirezionale dotato di due porte A e B utilizzante un protocollo di comunicazione SPI.
Ad ogni porta sono associati 8 bit: PortA e PortB; in modo tale da poter configurare il funzionamento in modalità 8-bit oppure 16-bit; dei quali, inoltre, è possibile configurare singolarmente i bit come ingressi oppure uscite.
Il protocollo SPI consente al sistema master, microntrollore, la configurazione dei registri del Port Expander, semplicemente scrivendo in essi; per esempio scrivendo nel registro IOCON.BANK si può configurare il dispositivo in modo tale da poter operare in modalità 8-bit oppure 16-bit; tramite i registri IODIRA/B si effettua la configurazione dei bit I/O. Inoltre il master ha la possibilità di leggere tutti i registri del "Port Expander".Tra i vari registri che lo contraddistinguono, fondamentali risultano i seguenti registri :

Per configurare i pin delle porte:
- IODIRA
- IODIRB

Per la configurazione generale del funzionamento:
- IOCONA
- IOCONB

Per la lettura delle porte:
- GPIOA
- GPIOB

Per la scrittura delle porte:
- OLATA
- OLATB

Inoltre, per quanto riguarda il trasferimento dei dati ho optato, per l'esempio che seguirà, in secondo articolo, per la modalità sequenziale (scrivendo IOCON.SEQOP=0).

Per la realizzazione del software di gestione del dispositivo farò uso del compilatore MikroC versione 6.0 della Mikroelettronika.
L'operazione di scrittura , avviene inviando dal master all' integrato, lo "slave address" (costituito da 4 bit fissi e 3 bit da stabilire (A0- A1-A2) che definiscono l' indirizzo hardware del dispositivo) ed il bit che indica la tipologia del comando R/W (0 indica la scrittura), come mostrato in figura 2 (che costituiscono il byte di controllo in quanto MCP23S17 è un SPI slave); questo comando costituisce l' opcode al quale devono essere aggiunti l' indirizzo del registro, come mostrato in figura 1, sul quale si vuole operare ed almeno un byte di dati. Analogamente si procede per il comando di lettura, in cui bisogna soltanto cambiare il bit che indica la tipologia del comando: R/W (1 indica la lettura).
Nel nostro caso, utilizziamo una modalità sequenziale, per cui parliamo di operazioni sequenziali di lettura e scrittura, nelle quali il master trasmette il successivo byte puntato dal puntatore di indirizzo fin

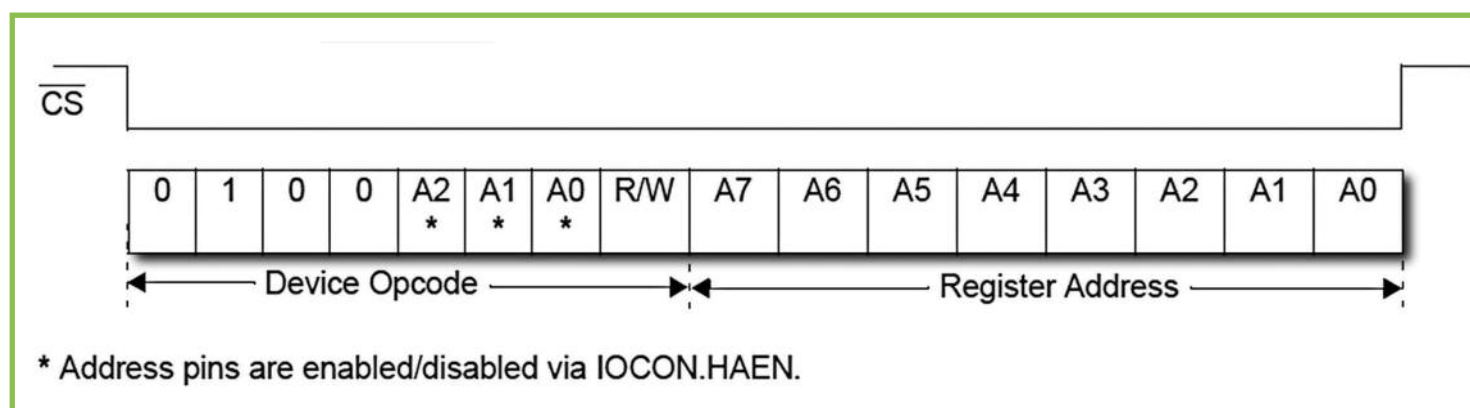



Figura 1- Indirizzamento di un registro.

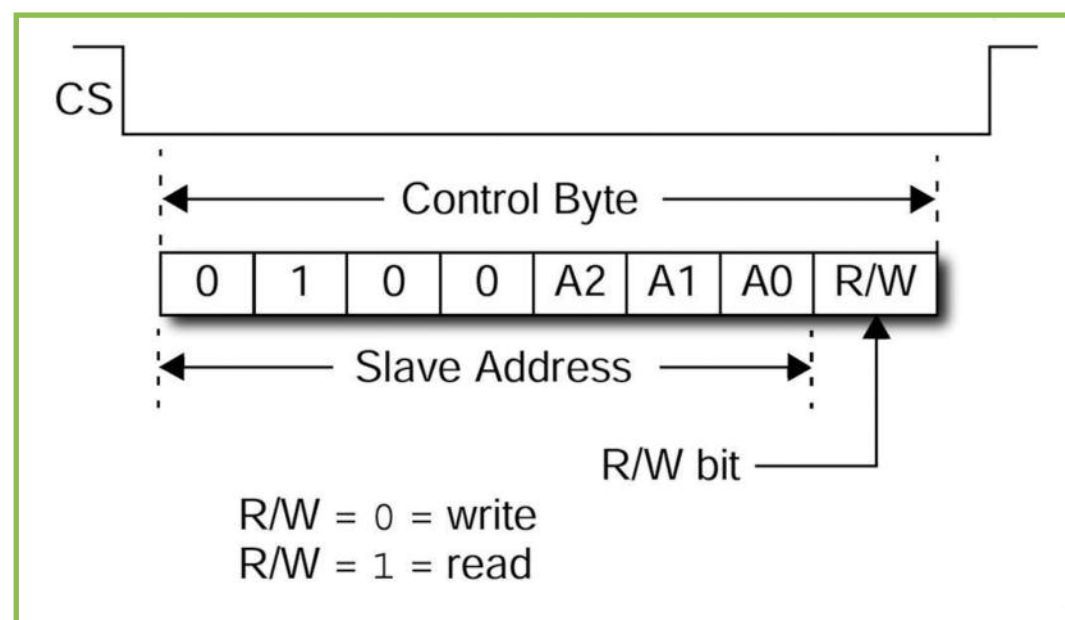



Figura 2- Definizione dell'opcode.

SUMMARY OF REGISTERS ASSOCIATED WITH THE GPIO PORTS (BANK = 1)

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	01	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	02	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPPUA	06	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPIOA	09	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	0A	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
IODIRB	10	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLB	11	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENB	12	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPPUB	16	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPIOB	19	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATB	1A	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

Tabella 1- Registri di controllo (IOCON.BANK=1)

quando la sequenza non termina, ovvero fin quando il master non pone il CS ad un livello logico alto, in questo caso il puntatore, dopo aver puntato l' ultimo registro, ritorna all' indirizzo iniziale.

I registri del Port Expander, sono raggruppati in due tabelle, le quali differiscono per la configurazione del bit BANK, appartenente al registro IOCON. In particolare:

• Se Bank=1 ,come in tabella 1,avremo da un lato l'insieme dei registri associati alla porta A, dall'altro i registri associati alla porta B.

• Se Bank=0,come mostrato in tabella 2, i registri A/B sono accoppiati.

Di queste due tipologie, la prima è usata nel caso in cui si opera con uno solo dei registri (A oppure B); mentre la seconda, se si opera utilizzando entrambi i registri (A e B), in particolare per le applicazioni dove si deve ottimizzare al massimo la velocità di trasmissione.

SUMMARY OF REGISTERS ASSOCIATED WITH THE GPIO PORTS (BANK = 0)

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

Tabella 2- Registri di controllo (IOCON.BANK=0)

Nell'applicazione che realizzerò in un secondo articolo utilizzerò il bit Bank uguale a "0".

Il Registro di direzione di porta (I/O Direction Register): consente il controllo della direzione dei dati, mediante la configurazione degli 8-bit come ingressi oppure uscite, come mostrato in figura 3. In particolare, quando un bit è posto a "1", il pin corrispondente è definito come ingresso, quando invece un bit è posto a "0", il pin corrispondente è definito come uscita.

Il Registro di configurazione generale (IOCON): contiene i bit per la configurazione generale dell' integrato, riportati nella figura 4.

In particolare, consente di definire come i registri vengono indirizzati (cioè quale delle due tabelle, precedentemente mostrate, si vuole utilizzare); si può stabilire di connettere oppure non connettere i pin INTA/B (interruzioni); consente di abilitare la modalità sequenziale di indirizzamento op-

Figura 3- Registro IODIR.

IOCON – I/O EXPANDER CONFIGURATION REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—
bit 7							bit 0

bit 7 **BANK:** Controls how the registers are addressed
1 = The registers associated with each port are separated into different banks
0 = The registers are in the same bank (addresses are sequential)

bit 6 **MIRROR:** INT Pins Mirror bit
1 = The INT pins are internally connected
0 = The INT pins are not connected. INTA is associated with PortA and INTB is associated with PortB

bit 5 **SEQOP:** Sequential Operation mode bit
1 = Sequential operation disabled, address pointer does not increment
0 = Sequential operation enabled, address pointer increments

bit 4 **DISSLW:** Slew Rate Control for SDA output
1 = Slew rate disabled
0 = Slew rate enabled

bit 3 **HAEN:** Hardware Address Enable for the MCP23S17 (**not used for the MCP23017**)
1 = Enables hardware address pins
0 = Disables hardware address pins (the device opcode becomes '0' for A2, A1 and A0. Pins must be externally biased regardless of HAEN setting)

bit 2 **ODR:** Configures the INT pin as an open-drain output
1 = Open-drain output (overrides the INTPOL bit)
0 = Active driver output (INTPOL bit sets the polarity)

bit 1 **INTPOL:** Sets the polarity of the INT output pin
1 = Active-high
0 = Active-low

bit 0 **Unimplemented:** Read as '0'

Legend:
R = Readable bit — W = Writable bit — U = Unimplemented bit, read as '0'
-n = Value at POR — '1' = Bit is set — '0' = Bit is cleared — x = Bit is unknown

Figura 4-Registro IOCON

GPIO – GENERAL PURPOSE I/O PORT REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
bit 7							bit 0

bit 7-0 **GP7:GP0:** Reflects the logic level on the pins <7:0>
1 = Logic high
0 = Logic low

Legend:
R = Readable bit — W = Writable bit — U = Unimplemented bit, read as '0'
-n = Value at POR — '1' = Bit is set — '0' = Bit is cleared — x = Bit is unknown

Figura 5- Registro GPIO

OLAT – OUTPUT LATCH REGISTER 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0
bit 7							bit 0

bit 7-0 **OL7:OL0:** Reflects the logic level on the output latch <7:0>
1 = Logic high
0 = Logic low

Legend:
R = Readable bit — W = Writable bit — U = Unimplemented bit, read as '0'
-n = Value at POR — '1' = Bit is set — '0' = Bit is cleared — x = Bit is unknown

Figura 6- Registro OLAT

pure no; si può abilitare un rate maggiore; si può decidere di abilitare oppure no i pin relativi all' indirizzo hardware; è possibile configurare il pin INT come uscita open-drain ,ed infine si può stabilire la polarità del pin INT di uscita se come attiva bassa o come attiva alta.

Il registro GPIO: leggendo questo registro leggiamo il valore presente sulla porta e scrivendo su questo registro viene modificato il registro di memoria in uscita (OLAT). La modalità di configurazione dei bit è riportata in figura 5.

In sostanza, questi bit riportano il livello logico sulla porta, che può essere attivo basso oppure attivo alto. Il Registro di memoria in uscita (OLAT): permette l'accesso ai pin di uscita, nel senso che leggendo questo registro, la lettura corrisponde ad una lettura dall' OLAT register e non dalla porta stessa. La modalità di configurazione dei bit è riportata in figura 6. In particolare, è utilizzato per scrivere il livello logico sui terminali di uscita, che può essere attivo alto oppure attivo basso. Ultimata la programmazione dei registri, per poter leggere e scrivere le porte dell' integrato, bisogna tener conto delle temporizzazioni relative al protocollo seriale del Port Expander: la scrittura seriale dei dati nel dispositivo, come mostrato in figura 7, avviene portando il CS ad un livello logico basso, dopo di ciò i dati saranno trasferiti alla porta interessata in corrispondenza dei fronti di salita del segnale SCK(serial clock per 8 impulsi) . In tal caso, la scrittura di ciascun singolo bit la si realizza in tre fasi: nella prima prepariamo il dato, nella seconda facciamo passare il clock da 0 a 1 (fronte di salita), nella terza facciamo passare il clock da 1 a 0 (fronte di discesa). Dopo i suddetti 8 impulsi di clock il CS viene riportato al livello logico alto. C' è da osservare, inoltre, che durante tutta la fase

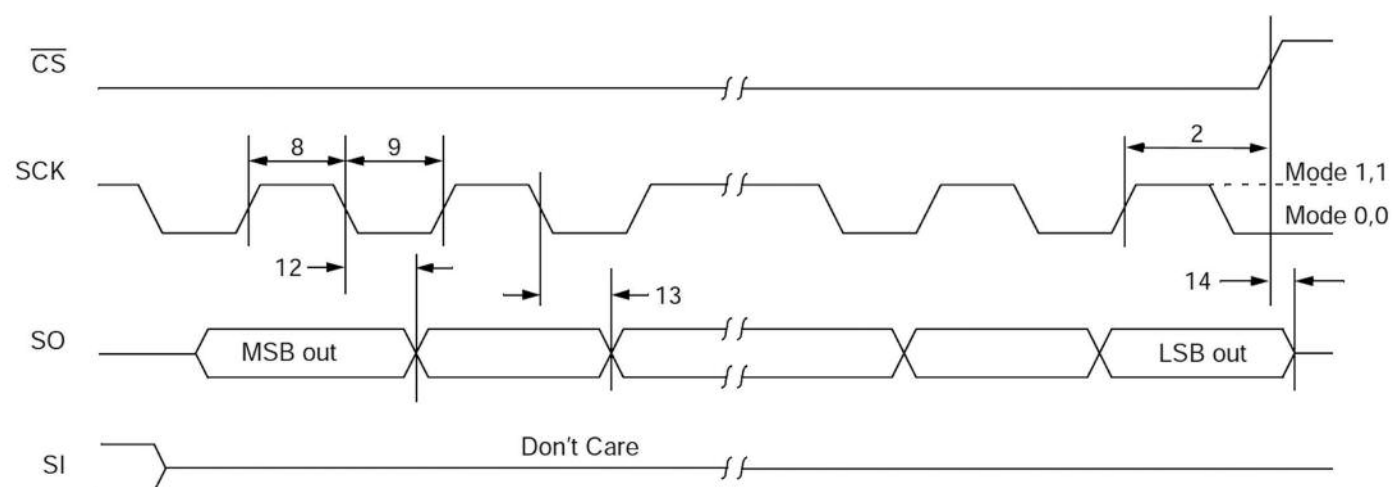



Figura 7- Temporizzazioni per la scrittura

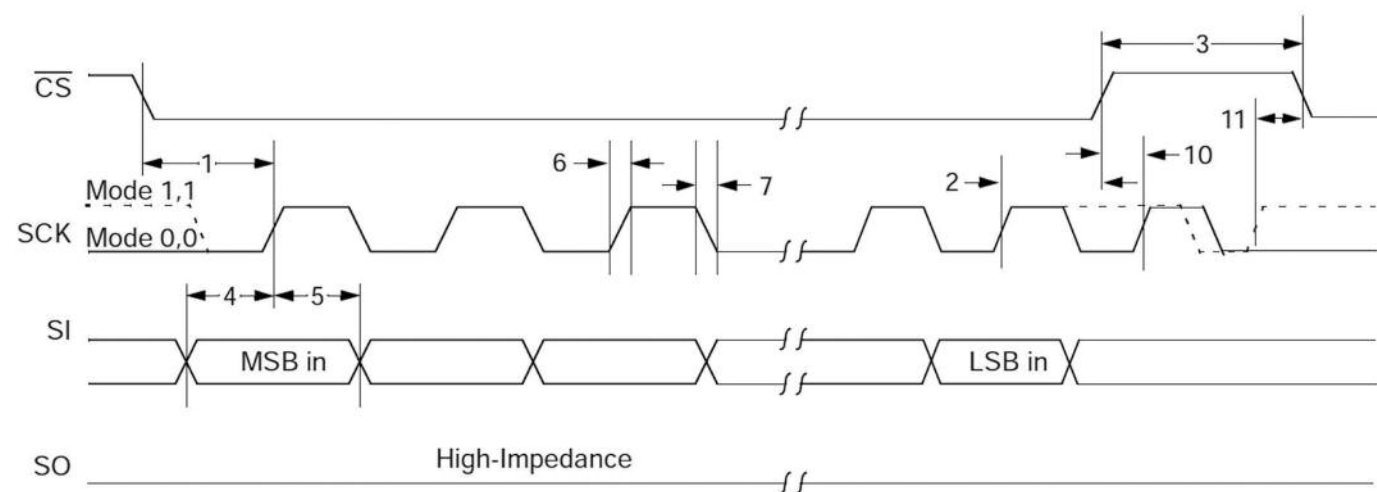



Figura 8- Temporizzazioni per la lettura

di scrittura dei dati il pin SO (dei dati in uscita) si trova nella condizione di alta impedenza, quindi inattivo. Nella fase di lettura, come mostrato in figura 8, il CS è portato ad un livello logico basso, dopo di che i dati saranno letti in concomitanza dei fronti di salita del SCK(serial clock per 8 impulsi). In tal caso, lettura del singolo bit lo si ottiene in tre fasi: nella prima fase si prepara la lettura, nella seconda fase portiamo il clock da 0 a 1 (fronte di salita), nella terza fase portiamo il clock da 1 a 0 (fronte di discesa).
Durante tutta la fase di lettura può avvenire anche il trasferimento dei dati, infatti il terminale SI (dei dati in ingresso) può trovarsi in un qualsiasi stato, e ciò non influenza in alcun modo la lettura. Ultimata la lettura il CS va riportato ad un livello logico alto.
Ora facciamo una succinta descrizione dell' integrato, in particolare il pin out dei terminali che lo caratterizzano, come mostrato in figura 9.

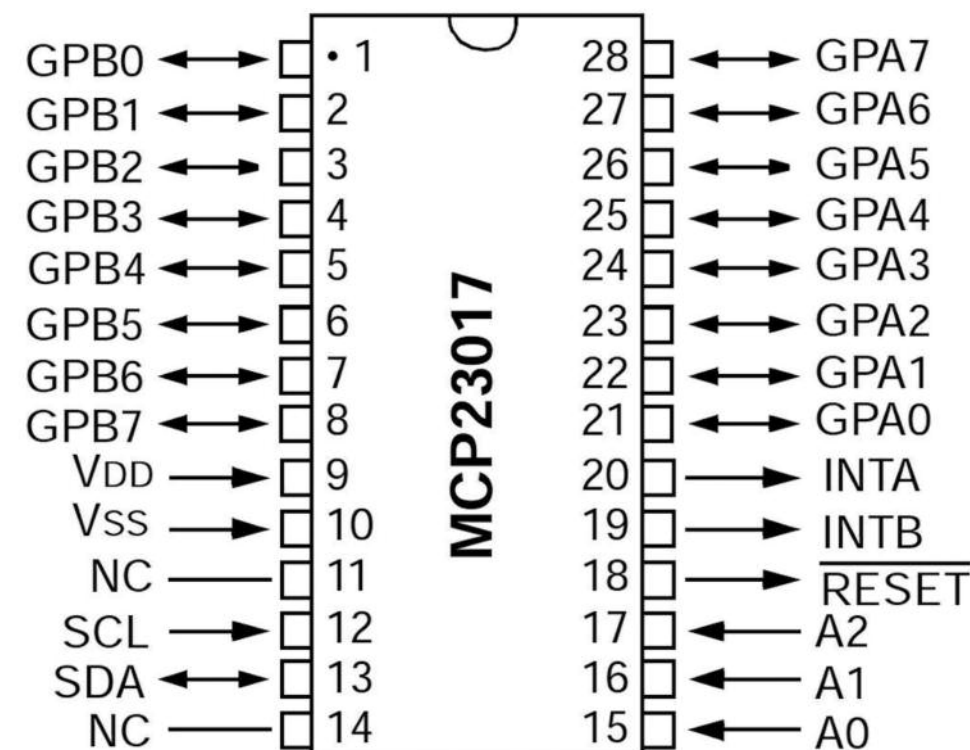



Figura 9- Descrizione dei Pin

GPBx- Pin I/O bidirezionali
Vdd- Tensione di alimentazione
Vss- Massa
CS- Chip Select
SCK- Serial Clock Input
SI- Dati in ingresso seriali **SO**- Dati in uscita serali
Ax- Pin di indirizzo hardware
RESET- Reset hardware (è attivo quando è posto ad un livello logico basso)
GPAx- Pin I/O bidirezionali.

Mediante i pin A0-A1-A2 di indirizzo fisico, si possono utilizzare contemporaneamente più dispositivi al massimo 8, in particolare il Port Expander è presente nella scheda Easy Pic7 e nel prossimo numero vedremo come utilizzarlo con alcune applicazioni pratiche.

di SILVANO BREGGION

HARDWARE

PLC CON INTERFACCIA USB

(parte terza)

Terza ed ultima parte del PLC in cui presenteremo l'interfaccia in visual basic e il progetto di una piccola scheda da collegare al nostro sistema in grado di misurare la temperatura con gli estremi del campo di misura diversificati per soddisfare le esigenze di tutti e l'uscita normalizzata 0-10 Volt.

A Nonostante il progetto del PLC sia di fatto concluso con il numero precedente della rivista, abbiamo pensato di integrarlo con un'interfaccia USB e un programma scritto in Visual Basic la cui grafica visualizza, in un'unica pagina, sia lo stato degli ingressi e delle uscite che il valore dei timer, dei contatori e degli ingressi analogici. Inoltre, essendo disponibili i file sorgente, chiunque potrà cimentarsi nel creare nuove interfacce sicuramente più accattivanti della nostra. Per i lettori esperti nel linguaggio di programmazione di Microsoft, ci auguriamo sia uno stimolo all'implementazione di nuove funzioni, come la possibilità di programmare il nostro PLC direttamente da PC.

Considerate le caratteristiche del progetto e la possibilità di gestire gli ingressi analogici, vi proponiamo in queste pagine una piccola schedina che misura la temperatura e la rende disponibile in un'uscita normalizzata 0-10 Volt. Niente di professionale o di estremamente complesso, al contrario un progettino molto semplice da costruire che vi permetterà di impratichirvi nella realizzazione di automazioni in cui coesistono, tra le varie funzioni, gli ingressi analogici non sempre facili da gestire.

Non pensate con questo di trovarvi tra le mani un oggetto puramente didattico, anzi è stato pensato per risolvere i problemi

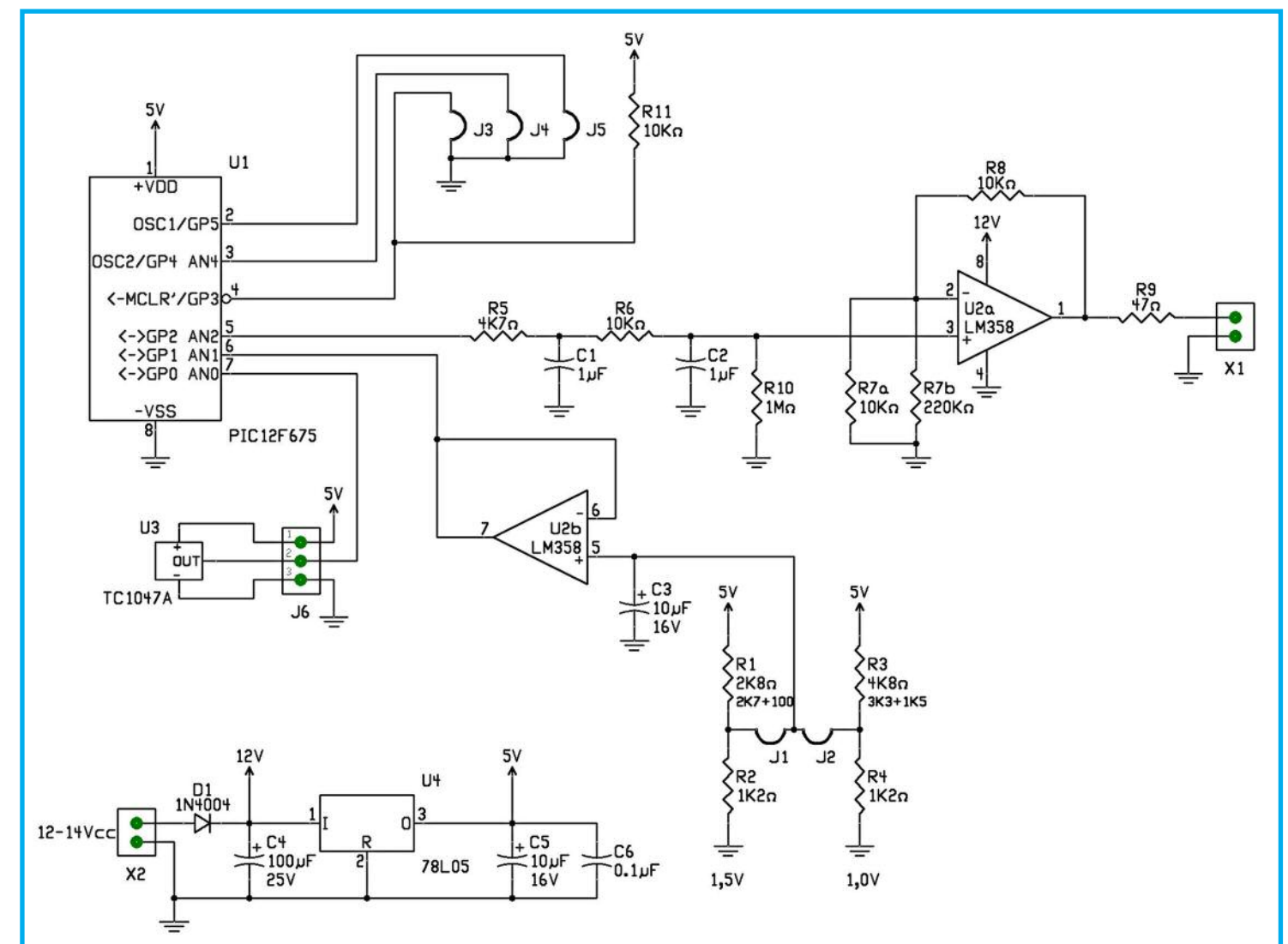



Figura 1: schema elettrico della sonda di temperatura

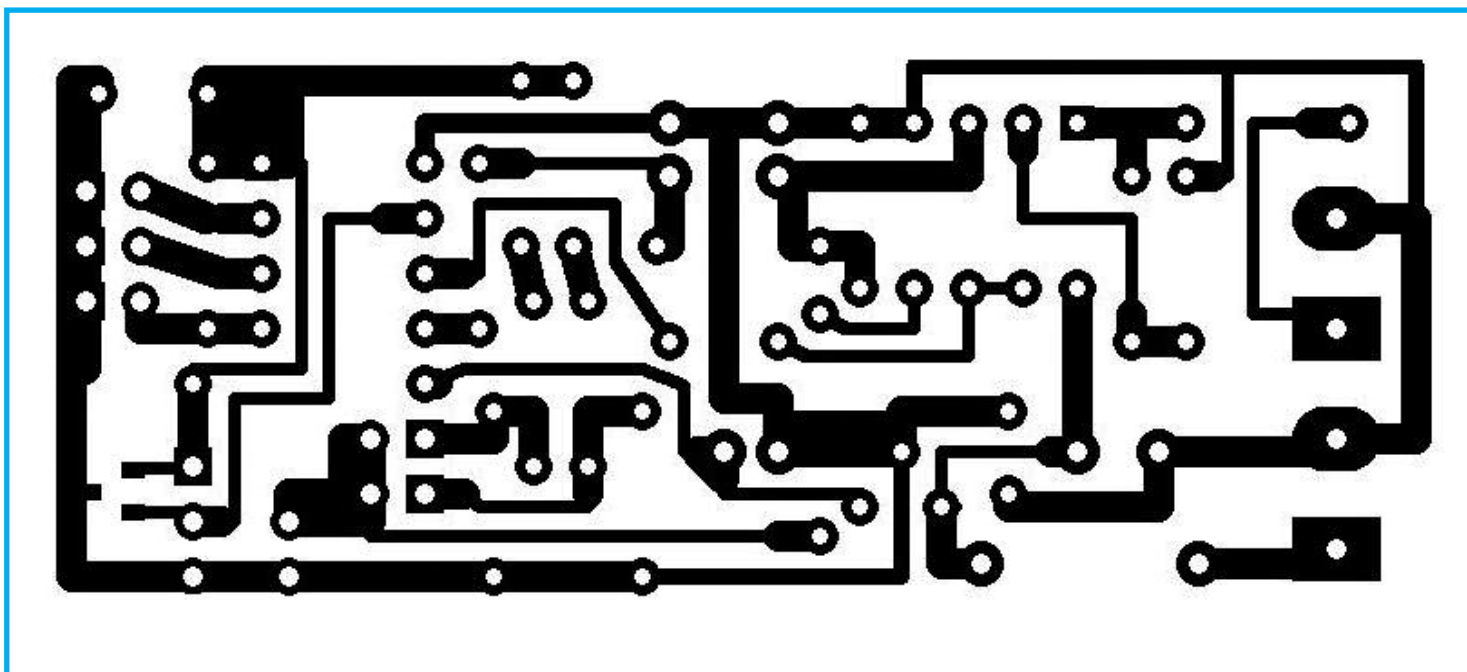



Figura 2: circuito stampato della sonda di temperatura

Foto 1: la sonda di temperatura

più disparati grazie alla possibilità di estendere o ridurre il campo di lavoro (span). Sono disponibili, semplicemente cambiando la disposizione di alcuni jumper, la selezione di limiti di temperatura sia verso l'alto che verso il basso come visualizzato nella tabella che segue:

0 – 50 °C
0 – 100 °C
-20 – 50 °C
-20 – 100 °C
-40 – 50 °C
-40 – 100 °C

Per realizzare una scheda che presentasse tali caratteristiche senza complicarci troppo la vita, abbiamo sfruttato le particolarità di un piccolo microcontrollore di casa Microchip. Tale scelta ci impone dei limiti per quanto riguarda la precisione nella conversione temperatura/tensione, ma offre la possibilità di sfruttare i moduli interni del PIC quali appunto il convertitore analogico digitale e il PWM usato in questo caso per generare in uscita una tensione proporzionale alla temperatura misurata. Il tutto con l'ausilio di pochissimi componenti esterni di tipo comune e di un doppio amplificatore operazionale che sicuramente avete a disposizione in un cassetto del vostro laboratorio.

SCHEMA ELETTRICO

Ci sono diversi modi per misurare una temperatura e diversi sono i tipi di sensori in grado di convertire la temperatura in una tensione, dalle resistenze a coefficiente negativo (NTC) ai PT100, dalle termocoppie agli integrati specializzati in tale mansione. Gli sperimentatori non più giovanissimi, ricorderanno senz'altro che l'unico modo per misurare una temperatura relativamente bassa, per esempio quella ambientale, si doveva far ricorso alle NTC. Il mercato le proponeva in diverse forme e fatture, di economicissime e di costosissime, tutte avevano in comune un unico difetto, non erano lineari. Col tempo, per nostra fortuna, i costruttori hanno cominciato a proporre dei circuiti integrati in grado di misurare la temperatura e convertirla direttamente in una tensione proporzionale. Attualmente gli NTC sono praticamente scomparsi dal mercato e quasi tutte le aziende che producono circuiti integrati hanno una loro linea di sensori tra cui quelli di temperatura. Tutta questa premessa per informarvi che anche Microchip dispone a catalogo di tutta una serie di sensori di temperatura, da cui abbiamo attinto uno dei modelli più economici, il TC1047A.
In contenitore SOT23 a tre piedini, è disponibile solo in formato SMD. La particolarità che ci ha colpito è stata la semplicità d'uso, una volta alimentato con una tensione che varia tra 2,5 e 5,5 Volt è pronto all'uso e l'uscita, a bassa impedenza, è in grado di pilotare l'ingresso analogico del PIC senza bisogno di interporre alcun buffer. Il campo di temperatura misurato dal sensore è molto lineare, tra –40 e +125°C e la tensione in uscita va da 100mV (-40°C) a 1,75 V (+125°C) con un rapporto temperatura/tensione di 10mV/°C.
Uscita dal sensore di temperatura è collegato direttamente all'ingresso analogico AN0 del PIC12F683 siglato U1. La presenza del connettore J6 può sembrare strana, in realtà abbiamo previsto il montaggio del sensore direttamente sul circuito stampato, ovviamente dal lato rame essendo il formato del sensore in SMD. A qualcuno potrà risultare comodo piazzare il sensore ad una certa distanza dalla scheda, in tale caso provvedete al montaggio di U3 in un piccolo circuito stampato che andrà raccordato al connettore J6 con uno spezzone di cavo possibilmente, anche se non necessariamente, schermato.
Il convertitore analogico/digitale del micro provvede a leggere la tensione in uscita dal sensore e la trasforma in bit che andranno poi processati dal modulo successivo, il PWM. Al fine di sfruttare la massima definizione possibile dai 10 bit del ADC, il riferimento alla massima tensione non è affidato ai 5 Volt di alimentazione bensì ad un partitore resistivo esterno vedi R1-R2 o R3-R4 bufferizzato da U2b usato come adattatore di impendenza.
Il partitore R1-R2 fissa la tensione di riferimento a 1,5 Volt mentre il partitore formato da R3-R4 fornisce 1,0 Volt definendo quindi, quali saranno le massime temperature selezionabili. La presenza dei due jumper J1 e J2 è necessaria per selezionare quale dei due partitori resistivi an-

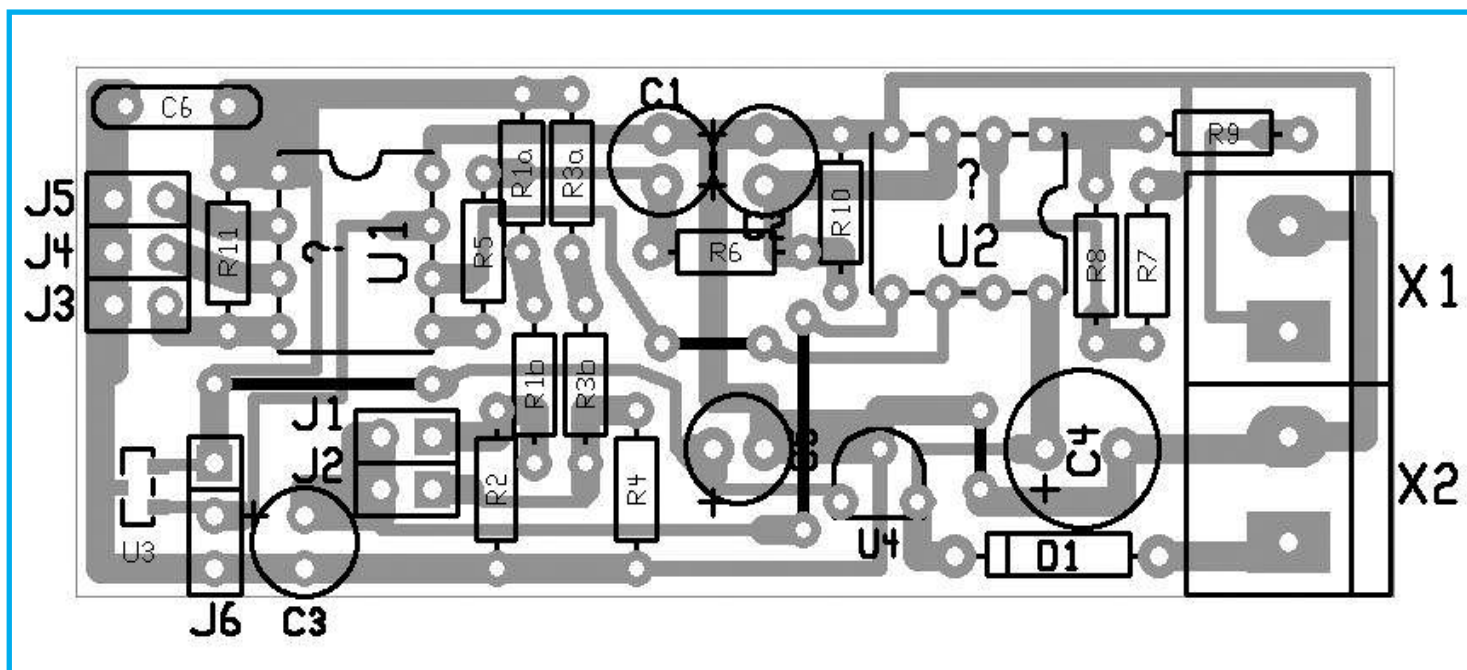



Figura 3: Piano di montaggio dei componenti

dranno a collegarsi all'ingresso non invertente di U2b e l'uscita di quest'ultimo all'ingresso analogico AN1 internamente collegato al riferimento esterno del ADC di U1. In particolare il jumper J1 collega il partitore resistivo formato da R1-R2 che imposta la massima temperatura misurabile a 100°C, mentre J2 collega il partitore R3-R4 che fissa la massima temperatura a 50°C. E' doveroso fare notare che le resistenze R1 e R3 in realtà sono formate ciascuna da due resistenze in serie in modo da ottenere il valore desiderato. Il circuito stampato prevede l'alloggiamento di ciascuna resistenza, per R1 trovate R1a e R1b, mentre per R3, R3a e R3b.
Al fine di fare capire al PIC quale delle due scale verso l'alto abbiamo selezionato, è stato inserito nello schema il jumper J3, se lasciato libero il micro gestisce il segnale analogico proveniente dalla sonda con la scala più ampia (100°C), mentre chiuso la scala è di 50°C.
Ricapitolando

- 50°C f. s.: J1 aperto, J2 chiuso, J3 chiuso
- 100°C f. s.: J1 chiuso, J2 aperto, J3 aperto

La presenza dei jumper J4 e J5 è necessaria per la selezione della minima temperatura come indicato dalla seguente tabella

- 0°C: J4 aperto, J5 aperto
- -20°C: J4 chiuso, J5 aperto
- -40°C: J4 aperto, J5 chiuso

La **tabella 1** riepiloga la posizione dei jumper per le scale di temperatura disponibili. Una volta informato della minima e della massima temperatura desiderabile, il PIC elabora il valore in bit proveniente dal ADC e lo trasforma in una percentuale da applicare al modulo PWM per poi variare il duty-cicle e ottenere in uscita una tensione proporzionale alla temperatura sfruttando la legge del valore medio.
Quest'ultima viene amplificata da U2a dopo essere stata filtrata da una doppia cella formata da R5-C3 e R6-C4. Il fattore di amplificazione di U2a in teoria dovrebbe valere G=2, ma le perdite dei filtri impongono una leggera correzione impressa dalla resistenza R7b in parallelo a R7a che favorisce un fattore di amplificazione di poco superiore a due.
La resistenza R9 ha la funzione di protezione da eventuali corto circuiti in uscita, pertanto può essere omessa se si pone attenzione al collegamento con il PLC.
Lo stadio di alimentazione è composto dal classico regolatore di tensione da 5 Volt, dai condensatori di livellamento e dal diodo D1 che protegge il circuito da dannose inversioni di polarità.
La tensione di alimentazione da applicare alla morsettiera X2, possibilmente stabilizzata, non deve scendere sotto i 12 Volt perché l'amplificatore U2, non essendo di tipo rail-to-rail, deve garantire l'escursione massima di tensione di 10 V. E' consigliabile non superare i 15 Volt per non surriscaldare il regolatore U4.

MONTAGGIO

Tutto il circuito trova posto in una basetta di piccole dimensioni, compresi i 4 ponticelli che ci evitano uno stampato a doppia faccia, la qui costruzione risulterebbe impegnativa. La presenza del piano di montaggio e l'esiguo numero di componenti non meritano alcun consiglio se non quello montare la sonda termica U3 per prima dal lato rame. Ovviamente se non vi interessa l'installazione in remoto della stessa.
E' possibile semplificare ulteriormente la scheda evitando il collegamento dei jumper da J1 a J5 sostituendoli con dei ponticelli in modo da selezionare la temperatura minima e massima desiderata, se il range non va cambiato nel tempo. Per le prime prove vi consigliamo la temperatura da 0 a 100°C e ottenere in uscita una tensione direttamente proporzionale alla temperatura senza bisogno di calcoli. Facciamo un semplice esempio, misurando ai morsetti di X1 un tensione di 2,13 Volt, la temperatura rilevata dalla sonda sarà di 21,3°C.
Per quanto riguarda il connettore J6, se avete saldato U3 direttamente allo stampato, ha poco senso montarlo.

COLLAUDO

Nulla di più semplice, date tensione al circuito con una tensione compresa tra 12 e

CAMPO	SPAN	J1	J2	J3	J4	J5
0-50°C	50		x	x		
0-100°C	100	x				
-20-50°C	70		x	x	x	
-40-50°C	90		x	x		x
-20-100°C	120	x			x	
-40-100°C	140	x				x

Tabella 1: Posizione dip switch in rapporto alla temperatura

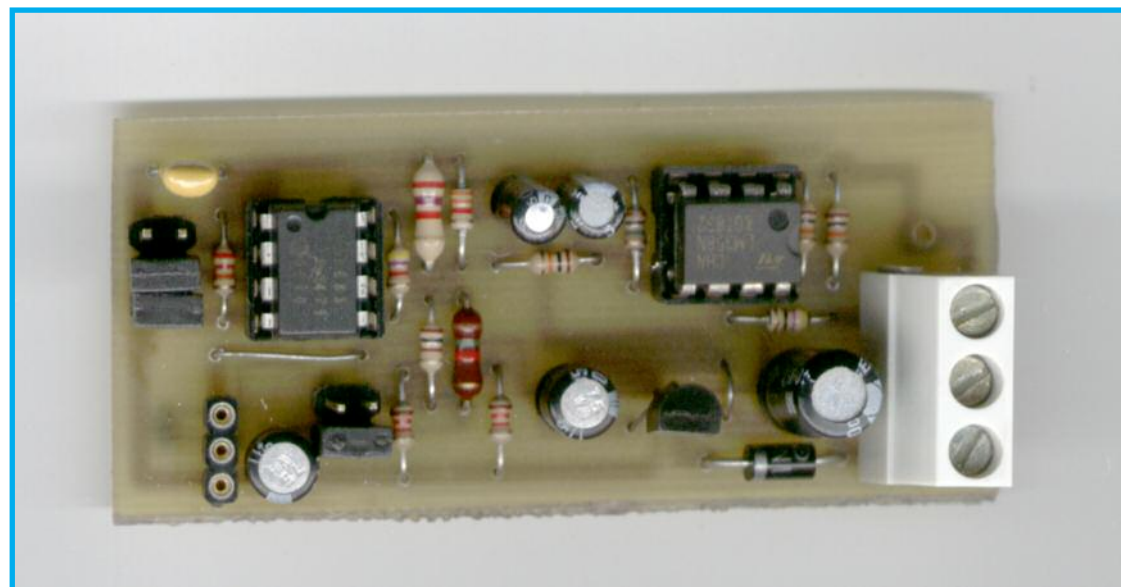



Foto 2: il convertitore

14 Volt e misurate la tensione ai morsetti di X1.
La relazione tra temperatura misurata dal sensore e la tensione in uscita è la seguente

Vout = (10 / span) * Ts

dove **Vout** è la tensione ai morsetti di X1, **span** è l'escursione della temperatura, mentre **Ts** è la temperatura rilevata dalla sonda espressa in °C. Difatti la formula inversa per ottenere la temperatura dalla tensione in uscita risulta

Ts = ((Vout / 10) * span) -T<0

se la minima temperatura è di 0°C viene ulteriormente semplificata

Ts = (Vout / 10) * span

in quanto **T<0** è la temperatura negativa della scala inferiore e vale 0 se la scala scelta è 0àC, **T<0** = 20 nella scala –20°C e **T<0** = 40 in quella da –40°C.
Appurata la completa funzionalità della scheda, non ci resta che collegarla ad uno degli ingressi analogici del PLC e iniziare a programmare.

INTERFACCIA USB

Una delle novità più importanti del nostro PLC è la possibilità di leggere con un PC allo stato delle proprie funzioni. Scartata l'idea di usare lo standad RS232 per la scomparsa di questa seriale da tutti i computer, la scelta è caduta, obbligatoriamente, sul USB ormai presente in tutte le macchine e considerata standard universale in quanto non esistono dispositivi da collegare al computer che usino un tipo di seriale diversa, presa RJ45 per collegamenti ethernet esclusa.
Il protocollo USB è molto complesso e difficilmente l'hobbista si cimenta nello scrivere il codice per implementarlo partendo da zero, qualche geniaccio del computer a parte e noi non facciamo eccezione.
L'azienda che produce i PIC, la Microchip, rende disponibile il codice completo per alcuni tipi di device tra questi anche la classe HID ottima per la nostra applicazione. La difficoltà incontrata dall'ambiente di sviluppo Mplab, piuttosto ostico da usare e il non facile utilizzo delle librerie, ci hanno orientato verso il MikroC, tra l'altro l'intero programma del PLC è scritto con questo IDE, scoprendo quanto sia semplice l'implementazione del USB grazie ad un Help comodo e funzionale arricchito da semplici esempi.
Vediamo assieme come usare la libreria MikroC per sviluppare una comunicazione USB prima dal lato PIC, poi dalla parte PC con Visual Basic.
Prima di tutto vanno dichiarate globalmente le due array che si occupano di contenere i dati da inviare al PC e viceversa.
unsigned char userWR_buffer[64], userRD_buffer[64];
Abbiamo conservato i nomi degli esempi, possono essere personalizzati a piacere, l'importante è riportarli uguali negli argomenti della funzione di inizializzazione della libreria. Quest'ultima deve essere richiamata all'avviamento del micro
HID_Enable(&userRD_buffer, &userWR_buffer);
Usando la funzione "copia e incolla" di windows, inserite nel vostro programma l'interrupt del modulo USB del PIC

```
void interrupt() {
    HID_InterruptProc();
}
```

Siete quasi pronti per inviare i vostri dati al PC copiandoli nell'array "userWR_buffer" con il comando
HID_Write(&userWR_buffer, 64);
i cui argomenti sono il buffer dati e il numero di Byte da inviare. E' importante notare che la funzione ritorna con "0" nel caso la trasmissione non sia avvenuta con successo, altrimenti con il numero di byte trasmessi.
Analogamente, quando il PIC riceve un pacchetto dal PC, lo parcheggia nell'array "userRD_buffer[64]" e sarà cura del programmatore controllare periodicamente se il buffer è pieno con la funzione
k = HID_Read();
dove "k" è una variabile qualsiasi e contiene il numero di byte ricevuti dal PC se la ricezione risulta corretta.
Per completare il programma dobbiamo dichiarare tutta una serie di variabili che sono necessari alla costruzione del pacchetto e alla parametrizzazione del modulo hardware USB del PIC. Nel caso volessimo personalizzare il collegamento con il nome nostro e del progetto, MikroC mette a disposizione un tool piuttosto intuitivo in grado di generare autonomamente il file da includere al firmware del PIC. Il tool si trova nel menù "Tools" alla voce "HID Terminal". Il tool propone di default la pagina "Terminal" utile nel testare il collegamento USB senza bisogno di sviluppare alcun programma dal lato PC. Per accedere al descrittore clicchiamo su "Descriptor" e cambiamo "Vendor Name" con il nostro nome e "Product Name" con il nome del progetto. Attenzione al campo "Report Lengt" deve contenere il valore "64" in ingresso e in uscita, mentre il campo "Bus power" permette all'host di offrire l'alimentazione di 5 Volt al device con una corrente massima di 500 mA. Dopo esserci assicurati di non avere cambiato il "VID" o il "PID" altrimenti il programma che gira sul PC non sarà in grado di riconoscere il device collegato alla presa USB, clicchiamo

Figura 4: installazione dei driver

sul pulsante “Save descriptor“ e salviamo il file generato dal tool nella stessa directory in cui si trova il programma che ci accingiamo a compilare.

Dal lato PC dobbiamo lanciare un programma in grado di riconoscere la nostra applicazione, salvare temporaneamente il pacchetto dati, tanto per intenderci quello contenuto dell’array “userWR_buffer“ che il PIC ha inviato al PC per mezzo della seriale USB, in un buffer per poi essere elaborato con le istruzioni del programma in funzione nel PC. Il programma è scritto con l’ambiente di sviluppo della Microsoft, Visual Basic 6 e per dialogare con la presa USB ha bisogno del file DLL (Dinamic Link Library) MCHID.dll che per svolgere correttamente il proprio lavoro deve essere copiato nella cartella “WINDOWS“. L’intera libreria di gestione della seriale USB non è farina del nostro sacco, ma è stata trovata in rete e sviluppata da un certo “Mecanique” a qui vanno tutti i nostri ringraziamenti.

Il bufer di ricezione usato nel programa in VB6 è stato chiamato

Dim BufferIn(0 To 64) As Byte

mentre quello di trasmissione, cioè dal VB6 al PIC è

Dim BufferOut(0 To 64) As Byte

Molto semplicemente la funzione in grado inviare il pacchetto dati dal VB6 al PIC è questa

Private Sub Refresh_Stato_PLC()

Come vedete il tutto è abbastanza semplice da gestire, basta fare un pò di attenzione nella stesura del programma e dopo qualche prova pratica, diventa davvero semplice.

Una volta ricevuto i pacchetto, è necessario filtrarlo per indirizzarlo al giusto utilizzo, per questo motivo abbiamo sfruttato i primi due byte dell’array alla codifica dello stesso.

Vediamo la parte del programma dal lato PIC **(listato 1)** e e dal lato VB **(listato 2)**.

E’ stato previsto un byte per la numerazione della scheda, nel caso specifico ne è presente solo una, sviluppi futuri potrebbero richiederne più di qualcuna e un secondo byte per indirizzare il pacchetto dati verso quella parte di programma in grado di analizzarli e gestirli. Ciò che può essere considerato superfluo per l’applicazione in oggetto, potrebbe diventare utile a chi si cimenta in modifiche dello stesso. Un esempio per tutti, sarebbe interessante programmare il nostro PLC direttamente dal PC, in questo caso diventa conveniente filtrare i pacchetti in transito sostituendo il valore di userWR_buffer[1] da ‘a’ a ‘p’ (programmazione) seguito dal buffer

LISTATO 1

```
void SendImageOut () {
 userRD_buffer[1] == ‘0’;      // Solo una volta
  userWR_buffer[0] = ‘1’;
   userWR_buffer[1] = ‘a’;
```

LISTATO 2

```
Public Sub OnRead(ByVal pHandle As Long) ‘ RICEVE STRINGA DA USB
 hidReadEx VendorID, ProductID, BufferIn(0)
  If (BufferIn(1) = Asc(“1”)) Then       ‘ Scheda #1
   If (BufferIn(2) = Asc(“a”)) Then    ‘VISUALIZZA VALORI FUN-
ZIONI
    Call Aggiorna_Stato_PLC
  …
```

Figura 5: il software in funzione

successivo contenente in numero del pacchetto inviato in quanto la memoria EEPROM del PIC contiene 256 byte e un pacchetto della classe HID ne trasporta un massimo di 64.

L'INTERFACCIA VB6

Lo scopo è quello di tenere sotto controllo tutte le funzioni del PLC in un'unica schermata e crediamo di esserci riusciti con in programma che vi proponiamo visibile nelle figure proposte in queste pagine. Nella parte alta della schermata possiamo osservare in bella mostra la parte più importante del PLC ovvero lo stato degli ingressi e delle uscite. E' vero che anche il display a bordo della scheda del PLC è programmato per la stessa funzione, ma con il programma in oggetto che gira sul PC oltre ad essere più appariscente risulta anche comodo. Le caselle rappresentanti gli ingressi e le uscite cambiano colore diventando gialli quando lo stato dei corrispettivi I/O assumono lo stato logico alto. E' sufficiente un colpo d'occhio allo schermo per capire cosa succede attorno al PLC, soprattutto se la nostra attenzione è rivolta allo stato di un particolare ingresso e alla risposta del programma immesso nella memoria del PLC a determinarne il corretto funzionamento seguendo l'andamento delle uscite.

Assieme agli ingressi e alle uscite è utile monitorare lo stato dei marker, a questo scopo sono stati aggiunti nel pannello virtuale anche quest'ultimi.

Subito sotto al gruppo appena menzionato, troviamo gli otto timer i quali riportano il tempo impostato, la grandezza del tempo, ovvero se si tratta di secondi o minuti e, infine, li conteggio in tempo reale.

La funzione che segue visualizza i quattro contatori attraverso tre finestre per ognuno di essi e visualizzano il valore da raggiungere nella prima, segue quello conteggiato in tempo reale ed infine il peso assegnato ad ogni impulso conteggiato.

Con l'ultima parte dell'interfaccia inerente agli ingressi analogici, terminiamo la descrizione ricordando che i valori analogici possono essere mostrati nella forma 0,00 - 10,00 Volt con la versione 1.1 e nel formato 00,0 – 100,0 nella versione 1.0. La differenza tra queste due versioni è tutta qua, la 1.1 è ottima per visualizzare la tensione applicata ai morsetti del PLC, mentre l'altra è ottimizzata per la misura della temperatura.

E SE IL PROGRAMMA NON GIRA

Nonostante Visual Basic sia l'ambiente di sviluppo della Microsoft e si presuma che gli eseguibili compilati siano pienamente compatibili con i sistemi operativi della stessa casa, potrebbe succedere che il programma del PLC si rifiuti di girare e, al contrario, una volta lanciato l'eseguibile compaia un messaggio a monitor che avvisa che nel computer non e' installato il file "msstdfmt.dll". Nel nostro PC con Windows XP abbiamo agito nel modo se-

Elenco componenti convertitore	
R1	2,7 KΩ + 100 Ω
R2	1,2 KΩ
R3	3,3 KΩ + 1,5 KΩ
R4	1,2 KΩ
R5	4K7 KΩ
R6	10 KΩ
R7	220 KΩ // 10 KΩ
R8	10 KΩ
R9	47 Ω
R10	1 MΩ
R11	10 KΩ
C1	1 µF 16 V
C2	1 µF 16 V
C3	10 µF 16 V
C4	100 µF 25 V
C5	10 µF 16 V
C6	0,1 µF
D1	1N4004
U1	PIC12F675
U2	LM358
U3	TC1047A
U4	78L05

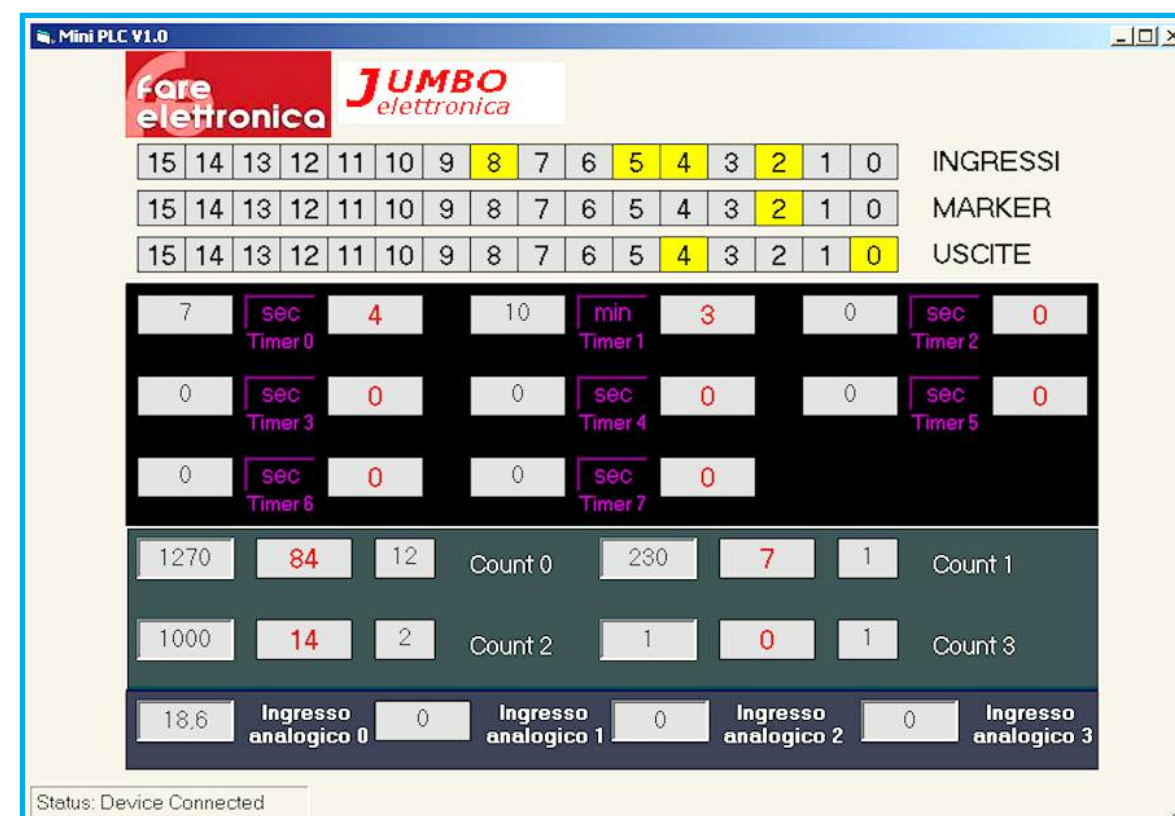



Figura 6: il software in funzione

Figura 6: il software in funzione

guente: copia/incolla del file “msstdfmt.dll“ che trovate nel pacchetto scaricato dalla rivista nella cartella “Support”, in c:\windows\system32, cliccate sul pulsante START e selezionate ESEGUI. Dalla finestra digitate CMD e cliccate su OK. Dalla finestra dei comandi digitate “regsvr32 msstdfmt.dll“ quindi ENTER. Se tutto è andato a buon fine appare un messaggio con l’indicazione che la registrazione è riuscita. Potrebbe essere necessario copiare e registrare altri file.dll, in questo caso dovete scaricarli da internet e procedere allo stesso modo. Non dovrebbe essere necessario il riavvio del computer.
Se siete stati costretti ad installate uno o più file.dll, con tutta probabilità sarà necessario installare e registrare il file “comdlg32.ocx“. Procedete con il copia/incolla del file “ comdlg32.ocx “ nella solita cartella c:\windows\system32, cliccate su START, selezionate ESEGUI, quindi digitate “regsvr32 %Systemroot%\System32\comdlg32.ocx“ e cliccate su OK. Un messaggio avviserà della riuscita della registrazione. A questo punto il programma del PLC come qualsiasi altro programma in VB, deve avviarsi. Nei PC che girano sotto Windows Professional, tipicamente quegli utilizzati negli ambienti industriali, non c’è bisogno di installare alcun file aggiuntivo. Evidentemente hanno qualcosa in più della versione “Home”.

Prossimamente

La pneumatica con il cubloc

Un esempio di applicazione alla pneumatica dei microcontrollori con l'impiego del PC per l'elaborazione e la memorizzazione dei dati.

Costruiamo un generatore di onda quadra

Realizziamo assieme un semplice generatore di segnale ad onda quadra, utile in tante occasioni. Esso è capace di produrre segnali a bassa ed alta frequenza.

Attivazione temporizzata di un carico

Una semplice ed utile realizzazione che alla pressione di un pulsante attiva un carico e lo disattiva dopo alcuni secondi.

Indagine sui lettori

Aiutaci a conoscerti meglio!

Con il tuo aiuto riusciremo a offrirti una rivista sempre più in linea con le tue aspettative.

Compila online il questionario all'indirizzo www.farelettronica.com/survey

Per ringraziarti **per il tuo tempo e la tua cortesia, ti invieremo** gratuitamente **un bellissimo** eBook **del valore di** 14,52 euro!

ElettroQuiz n. 335/336

FACILE

Al circuito di figura è applicato un segnale sinusoidale di ampiezza 2V e frequenza 1KHz. Quali delle seguenti affermazioni sono vere?

a) Vout è una sinusoide di ampiezza 2V e frequenza 1KHz;
b) Vout è un'onda quadra;
c) Vout è un segnale pressoché nullo;
d) Vout è una sinusoide di ampiezza 1V e frequenza 1KHz.

DIFFICILE

Dato il filtro di figura, si determini la frequenza di taglio e il valore della tensione di uscita quando Vin è un segnale in continua e quando Vin è un segnale sinusoidale di frequenza 1MHz.

IN PALIO: Maker card

Le risposte ai quiz "facile" e "difficile" vanno inviate esclusivamente compilando il modulo su www.farelettronica.com/eq specificando la parola chiave "**Bode**".

Le risposte e i vincitori (previa autorizzazione) sono pubblicati alla pagina www.farelettronica.com/eq a partire dal 15 del mese successivo alla pubblicazione sulla rivista.

A tutti i partecipanti verrà assegnato un buono sconto del 10% (validità 3 mesi dalla data di assegnazione) utilizzabile per un prossimo acquisto su **www.ieshop.it**

di GIOVANNI DI MARIA

RASPBERRY PI
ACQUISIZIONE DEI SEGNALI DIGITALI

Tecniche di programmazione e metodi per la rilevazione e l'acquisizione dei segnali digitali esterni, con il Raspberry Pi

L'articolo ha lo scopo di porre le basi iniziali alla programmazione delle porte di ingresso del Raspberry Pi. Si tratteranno concetti molto semplici ed elementari, al fine di cominciare con gradualità il processo didattico sul sistema.

INPUT DIGITALE

Un ingresso digitale è un terminale al quale possono essere applicati solo due differenti differenze di potenziale. Solo se saranno rispettati correttamente i due livelli di tensione consentiti, il sistema potrà riconoscerli ed agire di conseguenza. In tutti gli altri casi il sistema si comporterà in modo inprevisto.
Nel Raspberry Pi, il metodo di acquisizione avviene secondo la logica positiva, con una tensione di 3,3V per il livello logico alto (H) e una tensione di 0V (rispetto a massa) per un livello logico basso.
E' compito del progettista rispettare il livello massimo consentito, per non rischiare di distruggere la scheda. Quindi non collegate alcuna tensione TTL a 5V, poiché il Raspberry Pi non li tollera affatto. Esso non dispone di alcun controllo o di protezione contro le sovratensioni.

LA TASTIERA

Una grande ciliegia sulla torta è rappresentata dalla tastiera. Essa, da sola, con i suoi 102 e più tasti, rappresenta una immensa unità di input e, soprattutto, non va ad occupare preziosi ingressi, che non sono poi così numerosi.
Una tastiera dunque può essere utile in tutte quelle applicazioni dove il software interagisce con l'utente finale. Dal momento che la rilevazione della pressione dei tasti è affidata al software, possono essere previste tutte le casistiche possibili ed immaginabili.
In aggiunta, una tastiera possiede un suo buffer, un suo sistema di repeat automatico dei tasti ed anche una modalità molto avanzata di antirimbalzo.

LE PORTE DI I/O

Oltre alla tastiera, il Raspberry Pi dispone di alcune porte di input digitale che possono essere utilizzate come ingressi digitali.

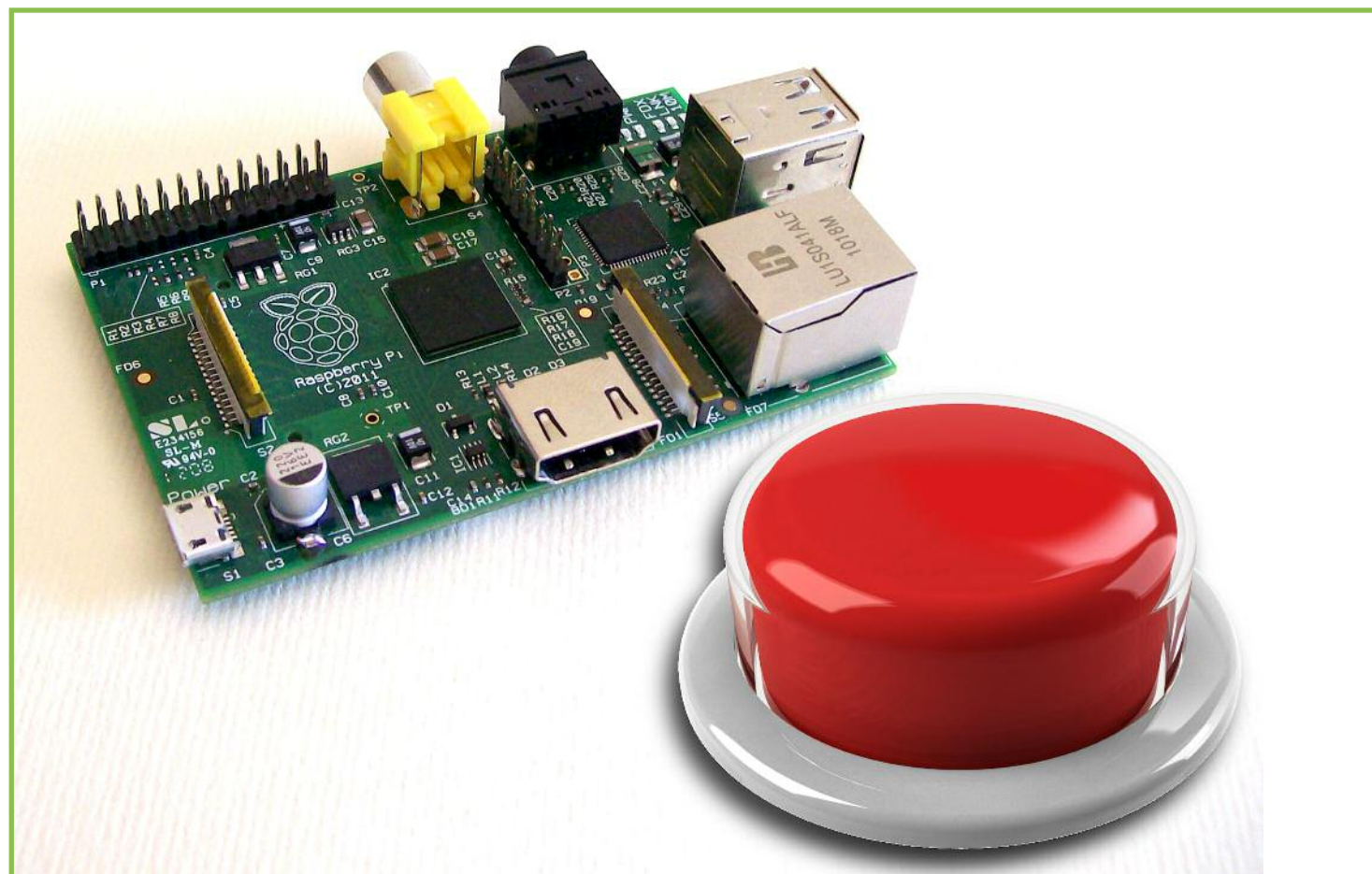



Figura 1: La tastiera è, da sola, una efficiente unità di input per il Raspberry Pi

Gli ingressi sono localizzati su una porta d'espansione formata da 26 pin, disposti in 2 file di 13 pin, con passo di 2,54 millimetri. Tale porta è contrassegnata con "P1" sulla scheda.
Il programmatore ha a disposizione ben 17 porte, utilizzabili quali ingressi e uscite, ovviamente con uso anche promiscuo. In modalità ingresso, tali porte, ricordiamolo nuovamente, accettano la tensione di 3,3V per il livello logico alto e 0V per il livello logico basso. Alcune di esse possono essere utilizzate anche per implementare altre funzionalità, come SPI, PWM, I2C e così via. Sono in progetto futuri articoli sulla rivista che le illustreranno in maniera approfondita.
I nomi di queste porte, ai fini della loro programmazione, sono i seguenti:

- GPIO0
- GPIO1
- GPIO4
- GPIO17
- GPIO21
- GPIO22
- GPIO10
- GPIO9
- GPIO11
- GPIO14
- GPIO15
- GPIO18
- GPIO23
- GPIO24
- GPIO25
- GPIO8
- GPIO

COME CONFIGURARE UNA PORTA IN INGRESSO

La tecnica è simile a quella vista nella rivista precedente, riguardo la configurazione delle uscite. Dal momento che gli ingressi sono "visti" come dei files virtuali, occorre aprire e registrare alcune informazioni. Di seguito sono riportate le fasi da seguire, per portare a termine le operazioni di creazione, settaggio, lettura e chiusura di una porta in input.
Gli esempi che seguono sono previsti per alcuni linguaggi, come ad esempio il C e Harbour. Tuttavia, compresa bene la filosofia di funzionamento, si può utilizzare qualunque linguaggio di programmazione previsto dal Raspberry Pi.

Creazione fisica della porta

Questa fase va eseguita solo una prima volta, per creare, sul file-system del Raspberry Pi, un riferimento fisico del file virtuale della porta. In pratica occorre scrivere un valore, corrispondente al numero della porta GPIO desiderata, dentro il file descritto dal seguente path:

```
/sys/class/gpio/export
```

Così, ad esempio, se desideriamo riferirci alla porta 4 (GPIO4) dobbiamo scrivere all'interno del file visto sopra, con qualunque mezzo, semplicemente il valore 4. Normalmente è consigliabile di chiudere immediatamente il file, dopo tale scrittura. Ricordiamo che il file deve essere aperto in modalità scrittura.

Definizione della direzione

Il successivo passo è quello della definizione della direzione. In altre parole occorre specificare al sistema la funzionalità della porta: in Input oppure in Output. Se si vuole configurare la porta precedente come ingresso, occorre andare a "disturbare" un altro file virtuale:

```
/sys/class/gpio/gpio4/direction
```

specificando stavolta la directory con il numero della porta corrispondente. In tale file, occorre scrivere la stringa "in" per programmare la porta come input. Ricordiamo che, anche in questo caso, che il file deve essere aperto in modalità scrittura.

Lettura della porta

La lettura della porta va fatta all'interno del programma e le tempistiche ovviamente sono decisi dal programmatore. In ogni caso, lo stato dell'ingresso è depositato in tempo reale in un ulteriore file del file-system, localizzato nella directory:

```
/sys/class/gpio/gpio4/value
```

Figura 2: La porta P1 è utilizzabile come Input e Output

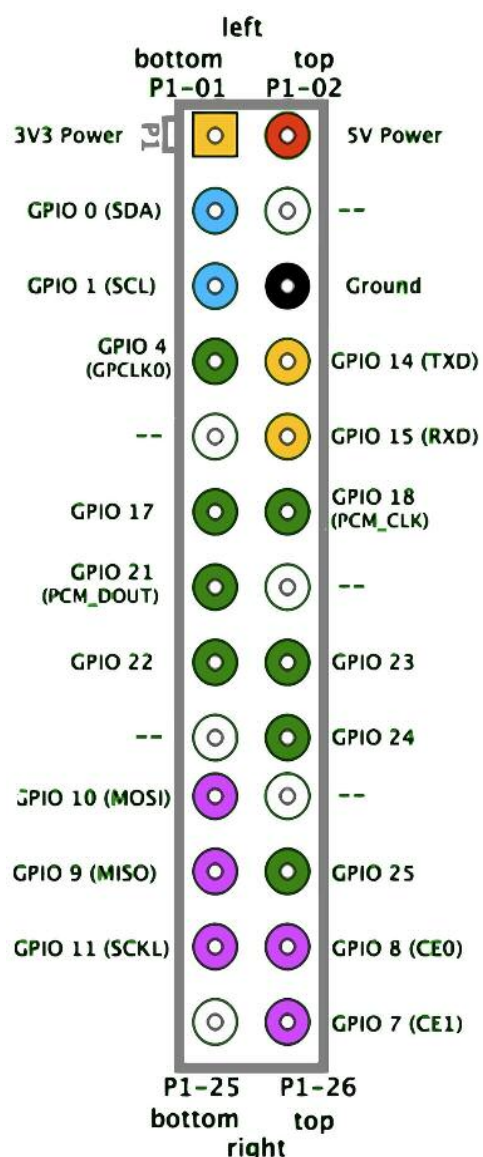



Figura 3: Disposizione delle porte GPIO (vista dall'alto)

Se il dato letto è 0, la porta si trova ad un livello logico basso. Se, viceversa la lettura restituisce il valore 1, la porta ha uno stato logico alto. Ricordiamo che il file deve essere aperto in modalità lettura. Anche in questo caso il file da leggere si trova all'interno della cartella con nome corrispondente alla porta in questione.

Rilascio della porta

Quando la porta non serve più e, in generale, quando il programma applicativo ha termine, è opportuno rilasciarla, scrivendo su un altro file del file-system del sistema. In pratica occorre scrivere un valore, corrispondente al numero della porta GPIO desiderata, dentro il file descritto dal seguente path:

```
/sys/class/gpio/unexport
```

Così, ad esempio, se desideriamo riferirci alla porta 4 (GPIO4) dobbiamo scrivere all'interno del file visto sopra semplicemente il valore 4. Il file deve essere aperto in modalità scrittura.

Riepilogo

Per rafforzare meglio la filosofia di funzionamento, peraltro semplicissima, riproponiamo qui un riepilogo immediato dei files virtuali da utilizzare per la programmazione di una porta in ingresso. L'esempio è riferito alla porta 4 GPIO4.

1. Creazione fisica della porta:
`/sys/class/gpio/export`
2. Definizione della direzione:
`/sys/class/gpio/gpio4/direction`
3. Lettura della porta:
`/sys/class/gpio/gpio4/value`
4. Rilascio della porta:
`/sys/class/gpio/unexport`

COME OTTENERE 3.3V

Dal momento che una porta di ingresso "esige" una tensione positiva di 3.3V, rispetto a massa, si deve impostare un sistema affinché possa erogare tale differenza di potenziale. Ovviamente non si tratta di una tensione critica, in quanto anche 3V o 2.8V contribuiscono positivamente a far riconoscere uno stato logico alto. Ovviamente tensioni più basse po-

Figura 4: Le porte GPIO (in verde)

trebbero costringere il Raspberry ad entrare in uno stato di "indecisione" logica.
Al fine di fornire alcune idee e soluzioni alternative, elenchiamo un insieme di metodi, al fine di ottenere e ricavare una tensione di circa 3V-3.3V, cui sottoporre l'ingresso:

- Se si dispone di una tensione di 5V, la si può abbassare con 3 diodi in serie;
- Si può usare direttamente la tensione di 3.3V prelevata dal pin 1 del bus GPIO;
- A seconda della fonte di alimentazione a disposizione, si può utilizzare un partitore resistivo con nodo centrale calcolato per generare 3.3V. Dal momento che la porta di ingresso è ad alta impedenza, le resistenze non devono dissipare molta potenza e possono avere, quindi, un valore ohmico relativamente elevato;
- Utilizzare un semplice regolatore di tensione LD1117V33 (TO-220). Questa soluzione è estremamente semplice perché utilizza solo due condensatori ed una resistenza;
- Utilizzare la classica configurazione a resistore + diodo Zener, con anodo a massa;
- Utilizzare un semplice regolatore di tensione LM317 (TO-220);
- Utilizzare 2 pile in serie da 1.5V;
- Ed altri ancora.....

PRIMO ESEMPIO: UN PULSANTE SU GPIO0

Il primo esempio è, come al solito, molto semplice ed ha lo scopo di rendere indolore ed efficace il primo approccio alla programmazione. Si tratta di una piccola applicazione che visualizza, sul monitor del TV, lo stato logico della porta GPIO0, configurata come ingresso. Il programma "gira" in modo continuo e veloce.
E' solo una versione didattica per far capire direttamente e in tempo reale, cosa avviene all'ingresso. Ricordiamolo sempre, la porta deve essere sottoposta ad una tensione di 3,3V se si intende inviare un segnale logico alto, altrimenti la si pone a massa, tramite una resistenza di pull-down.
Ricordiamo che per pilotare le porte GPIO occorre accedere nel sistema come root.

UN PULSANTE SU GPIO0 CON IL LINGUAGGIO C

Sicuramente farà piacere che uno dei tanti linguaggi di programmazione del Raspberry Pi è il linguaggio C. La usa universalità permette di apprenderlo immediatamente e di scambiare idee e informazioni praticamente con tutti gli utenti nel mondo. Vediamone dunque i dettagli logici e di programmazione, per poi commentarlo in maniera approfondita. Il listato, in sorgente C contiene un numero progressivo, in corrispondenza di ogni linea. Non lo si deve digitare in fase di editing, ma serve solo come riferimento per le spiegazioni di queste pagine.
Ecco di seguito le funzioni in dettaglio, riga per riga:

- La riga 01 ha il compito di includere i prototipi delle funzioni dello standard input e output;
- La riga 02 crea la funzione principale del programma, che sarà eseguita all'avvio del software;
- La riga 03 dichiara la variabile "h" come un puntatore a file. Essa sarà utilizzata per la gestione dei files del file-system affinché il programma possa accedere alle porte di input e di output;
- La riga 04 dichiara la variabile intera "ch". Essa conterrà il valore dello stato logico della porta di ingresso, prelevata da un opportuno file, in lettura;
- La riga 06 apre in scrittura il file "/sys/class/gpio/export" al fine di creare una nuova porta GPIO, con numero specificato alla prossima istruzione;
- La riga 07 scrive nel file appena aperto il

Figura 6: Acquisizione di un ingresso in corso (linguaggio C)

Figura 5: Connessione di un pulsante su GPIO0: schema elettrico

Figura 7: Acquisizione di un ingresso in corso (linguaggio Harbour)

valore 0. Ciò indica l'intenzione di gestire la porta GPIO0. Le altre porte seguiranno ovviamente diversa numerazione;

- La riga 08 chiude il file aperto in precedenza;
- La riga 10 apre in scrittura il file "/sys/class/gpio/gpio0/direction" al fine di stabilire la direzione dei dati della GPIO, tra input e output;
- La riga 11 scrive, nel file appena aperto, il valore "in". Ciò indica l'intenzione di tratta la porta GPIO0 con ingresso. Scrivendo invece il valore "out" si indica al sistema di considerare tale porta come uscita;
- La riga 12 chiude il file aperto in precedenza;
- La riga 13 introduce un "loop" infinito, ossia un blocco nel quale si ripetono "per sempre" le istruzioni software in esso contenute;
- La riga 15 apre in lettura il file "/sys/class/gpio/gpio0/value" al fine di leggerne lo stato logico della GPIO0;
- La riga 16 legge il contenuto del file, e quindi dello stato logico di GPIO0, memorizzandolo nella variabile intera "ch";
- La riga 17 stampa a video lo stato logico della porta GPIO0;
- La riga 18 chiude il file aperto in precedenza.

Compilazione del sorgente C

Si scriva il sorgente in un file, con estensione ".c", con il proprio editor preferito. Si consiglia di utilizzare il "vi", il più famoso e potente editor di Linux, anche se la sua funzionalità avviene da console.
Si compili dunque il listato con il compilatore gcc, in dotazione del sistema operativo. Basta scrivere al prompt di console il seguente comando:

```
gcc prova.c
```

Se il listato è esente da errore, il compilatore produce il file eseguibile (a.out), che l'utente avvierà con il comando:

```
./a.out
```

Lo schermo, all'avvio del software, mostrerà in rapida successione lo stato logico della porta GPIO0 e, premendo il pulsante ad essa collegato, cambierà il valore di tensione applicato sulla porta. Il televisore testimonierà in tempo reale tali cambiamenti di stato. Per concludere l'esecuzione occorre premere sulla tastiera i tasti <CTRL> <C>.

UN PULSANTE SU GPIO0 CON IL LINGUAGGIO HARBOUR

Harbour è un potente linguaggio di programmazione, orientato agli oggetti, molto semplice e dotato di strumenti per la programmazione veloce ed immediata di interfacce utenti (menù, navigazione sui databases, moduli di inserimento dati, ecc). Harbour è la moderna rivisitazione del famoso linguaggio Clipper, che per decenni ha dominato in tutto il mondo. L'implementazione di Harbour su piattaforma ARM per Raspberry Pi è una delle caratteristiche più eccitanti della programmazione di questo sistema. La filosofia del progetto e lo schema elettrico sono gli stessi per l'esempio visto in precedenza, con il linguaggio C. E' opportuno confrontare i due listati, al fine di scoprirne le differenze logiche e lessicali, proprie dei due linguaggi di programmazione. Anche in questo caso, le righe del programma saranno numerate progressivamente. Tali numeri non devono essere riportati in fase di codifica, ma servono unicamente come scopo di riferimento. Ecco di seguito la funzione in dettaglio, riga per riga:

LISTATO 1

```
00  /*——Acquisizione da pulsante. By Giovanni Di Maria—-*/
01  #include “stdio.h”
02  main() {
03     FILE *h;
04     int ch;

05     /*——CREA PORTA GPIO0———*/
06     h=fopen(“/sys/class/gpio/export”,”w”);
07     fprintf(h,”0”);
08     fclose(h);

09     /*———Porta GPIO0 in INPUT—————*/
10     h=fopen(“/sys/class/gpio/gpio0/direction”,”w”);
11     fprintf(h,”in”);
12     fclose(h);

13     while(1) {
14        /*———Apre e legge Porta GPIO0—————*/
15        h=fopen(“/sys/class/gpio/gpio0/value”,”r”);
16        fscanf(h,”%d”,&ch);
17        printf(“Stato GPIO= %d\n”,ch);
18        fclose(h);
19     }
20  }
```

Download

- La riga 01 introduce la procedura esecutiva main;
- La riga 02 dichiara le variabili locali "h" (handle del file) e cLettura (dato letto dal file);
- La riga 03 imposta il colore di background del video a blu con testo bianco;
- La riga 04 cancella lo schermo;
- La riga 06 apre in scrittura il file "/sys/class/gpio/export" al fine di creare una nuova porta GPIO, con numero specificato alla prossima istruzione;
- La riga 07 scrive nel file appena aperto il valore 0. Ciò indica l'intenzione di gestire la porta GPIO0. Le altre porte seguiranno ovviamente diversa numerazione;
- La riga 08 chiude il file aperto in precedenza;
- La riga 10 apre in scrittura il file "/sys/class/gpio/gpio0/direction" al fine di stabilire la direzione dei dati della GPIO, tra input e output;
- La riga 11 scrive nel file appena aperto il valore "in". Ciò indica l'intenzione di tratta la porta GPIO0 con ingresso. Scrivendo invece il valore "out" si indica al sistema di considerare tale porta come uscita;
- La riga 12 chiude il file aperto in precedenza;
- La riga 13 introduce un "loop" infinito, ossia un blocco nel quale si ripetono "per sempre" le istruzioni software in esso contenute;
- La riga 15 inizializza il buffer di lettura "cLettura" con un carattere;
- La riga 16 apre in lettura il file "/sys/class/gpio/gpio0/value" al fine di leggerne lo stato logico della GPIO0;
- La riga 17 legge il contenuto del file, memorizzandolo nella variabile intera "ch";
- La riga 18 chiude il file aperto in precedenza;
- La riga 19 visualizza a video lo stato della porta GPIO;
- Le righe 20, 21 e 22 controllano la pressione di un eventuale tasto <ESC> da tastiera, nel qual caso il ciclo termina;
- La riga 26 apre in scrittura il file "/sys/class/gpio/unexport" al fine di rimuovere una porta, con numero specificato alla prossima istruzione;
- La riga 27 scrive nel file appena aperto il valore 0. Ciò indica l'intenzione di eliminare, rilasciandola, la porta GPIO0. Le altre porte seguiranno ovviamente diversa numerazione;
- La riga 28 chiude il file aperto in precedenza;
- La riga 29 termina l'esecuzione della procedura main, con uscita dal programma.

Compilazione del sorgente Harbour

Si scriva il sorgente in un file, con estensione ".prg", con il proprio editor preferito. Si consiglia, anche in questo caso, di utilizzare il programma "vi", il più famoso e potente editor di Linux, anche se la sua funzionalità avviene da console.
Si compili dunque il listato con il compilatore di Harbour, da scaricare e installare separatamente. Allo scopo, bisogna scrivere al prompt di console il seguente comando:

```
hbmk2 test.prg -w3
```

Se il listato è esente da errore, il compilatore produce il file eseguibile (test), che l'utente avvierà con il comando:

```
test
```

LISTATO 2

```
00  //—Acquisizione da pulsante. By Giovanni Di Maria—
01  PROCEDURE main()
02     LOCAL h, cLettura
03     SET COLOR TO w/b
04     CLEAR SCREEN
05     //————-Crea porta GPIO0————
06     h = Fopen( "/sys/class/gpio/export", 1 )
07     Fwrite( h, "0" )
08     Fclose( h )
09     //———Definisce porta GPIO0 in INPUT——-
10     h = Fopen( "/sys/class/gpio/gpio0/direction", 1 )
11     Fwrite( h, "in" )
12     Fclose( h )
13     DO WHILE .T.
14        //—————Legge porta GPIO0————
15        cLettura = Space( 1 )
16        h = Fopen( "/sys/class/gpio/gpio0/value", 0 )
17        Fread( h, @cLettura, 1 )
18        Fclose( h )
19        ? "Stato GPIO0: " + cLettura
20        //———-Se premo tasto ESC, esce——-
21        IF Inkey() = 27
22           EXIT
23        ENDIF
24     ENDDO
25     //———-Rilascia porta GPIO0———
26     h = Fopen( "/sys/class/gpio/unexport", 1 )
27     Fwrite( h, "0" )
28     Fclose( h )
29     RETURN
```

Download

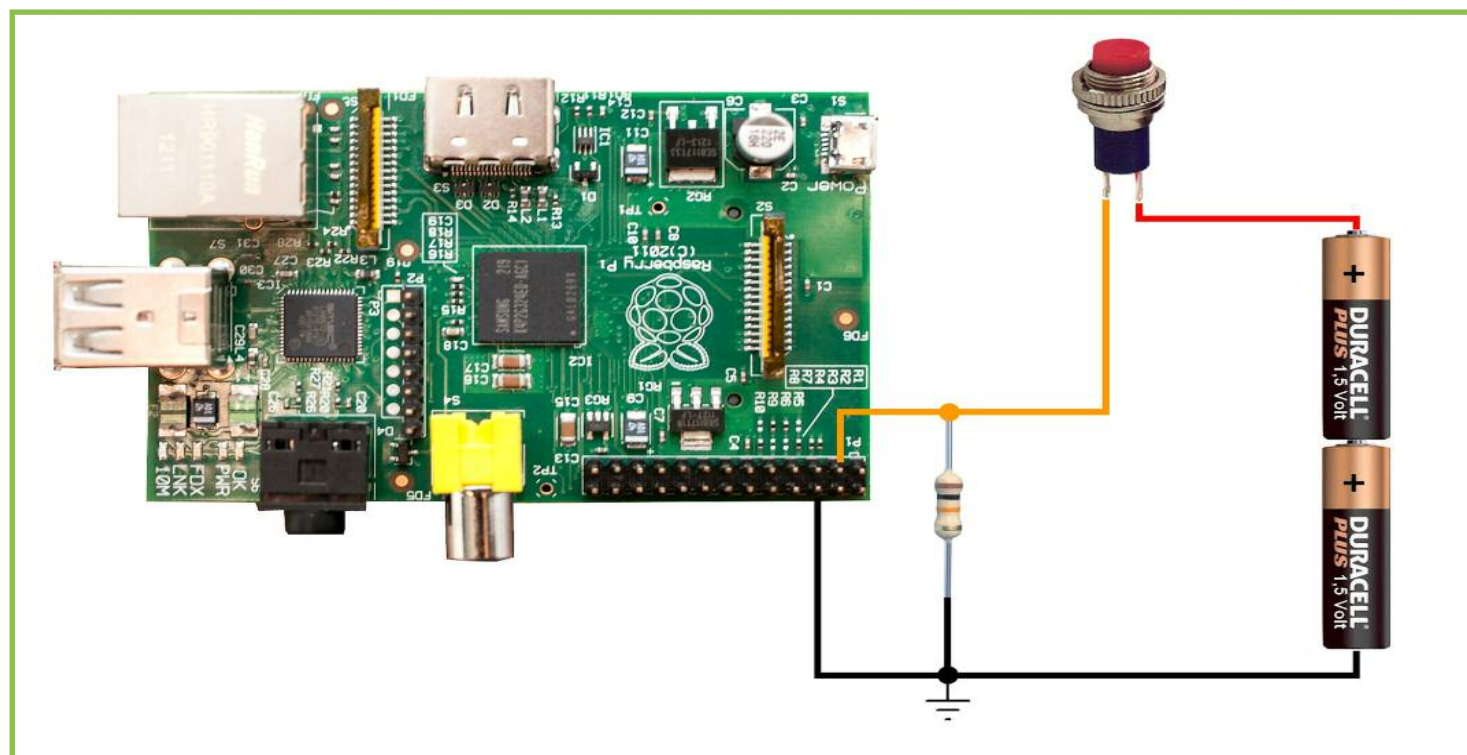



Figura 8: Connessioni fisiche del pulsante al Raspberry Pi

Anche in questo caso, lo schermo, all'avvio del software, mostrerà in rapida successione lo stato logico della porta GPIO0 e, premendo il pulsante ad essa collegato, cambierà il valore di tensione applicato sulla porta.
Il televisore testimonierà in tempo reale tali cambiamenti di stato.

CONSIDERAZIONI FINALI E CONCLUSIONI

Per programmare il Raspberry Pi si può utilizzare un grande numero di linguaggi di programmazione, come ad esempio python, perl, ruby, bash, c ed altri.
La scelta di Harbour è caduta poiché, tale linguaggio, è estremamente potente e, come vedremo in futuro, offre all'utente sistemi semplici per creare interfacce finali di alto livello, sistemi di temporizzazioni e gestione del video, altrimenti gestibili con molta fatica, con altri linguaggi. Fortunatamente il porting di Harbour per l'architettura del Raspberry funziona estremamente bene. L'utente è naturalmente libero di adottare il linguaggio di programmazione che più conosce e più soddisfa le sue esigenze.
Ricordiamo che per pilotare le porte GPIO occorre accedere nel sistema come root.
Come detto all'inizio, l'articolo costituisce un primo approccio, molto semplice, alla programmazione del Raspberry Pi, ed in particolare delle porte di ingresso. Non sono stati trattati argomenti un tantino più complessi e si è preferito rimanere ad un livello didattico piuttosto basso, ma efficace.
Nelle prossime puntate innalzeremo sicuramente il grado di difficoltà e presenteremo progetti ben più utili e complicati, che il lettore saprà affrontare con una maggiore cognizione di causa.

di ANTONIO GIANNICO

CORSO MIKROPASCAL PER PIC
INTERRUPT E TIMER

(parte ottava)

Prima di illustrare alcune applicazioni pratiche degli interrupt e dei timer è opportuno fornire le basi teoriche su queste importantissime risorse e funzionalità

Il concetto di interrupt è utilizzato soprattutto in informatica e nasce come risposta a problematiche legate alla gestione delle periferiche da parte di un processore.

In questo senso, un interrupt (o interruzione) è un segnale che indica il bisogno di attenzione da parte di una periferica. Con un tale segnale la periferica invia sostanzialmente una richiesta di servizio al sistema operativo ma la stessa richiesta può venire anche da un processo in esecuzione qualora si verifichino determinate condizioni. Per questo motivo, gli interrupt possono essere sia hardware che software. Gli interrupt hardware sono normalmente generati da dispositivi esterni alla CPU (generalmente dispositivi di I/O). Gli interrupt software sono invece originati da eventi di natura software legati ai processi in esecuzione.

Un interrupt comporta, in ogni caso, che il processore debba prontamente memorizzare lo stato del processo in esecuzione ed iniziare l'esecuzione di una routine che esegue il compito richiesto dall'interrupt. Servito l'interrupt, terminato cioè il servizio che ne ha portato alla risoluzione, il processore riprende l'esecuzione del processo interrotto precedentemente. Le situazioni in cui in un calcolatore possono verificarsi degli interrupt sono molteplici; si verifica un interrupt, per esempio, se:

- un processo tenta di eseguire operazioni non valide come una divisione per zero;
- un dispositivo di I/O informa la CPU che è disponibile ad inviare o ricevere dati;
- in occasione del debugging di un codice.

Questi tre esempi possono essere compresi in maniera intuitiva; in realtà gli interrupt che possono interessare una CPU sono molto più numerosi di quanto si possa pensare, sia per tipologia che per la frequenza con la quale si manifestano.

L'integrazione, nell'architettura di una CPU, di interrupt apre infatti le porte ad una gestione particolarmente efficiente e dinamica di eventi sia interni che esterni, sia hardware che software che necessitano, per loro natura, di urgente attenzione. In genere si tratta di eventi la cui gestione perderebbe di efficacia nel caso in cui fosse ba-

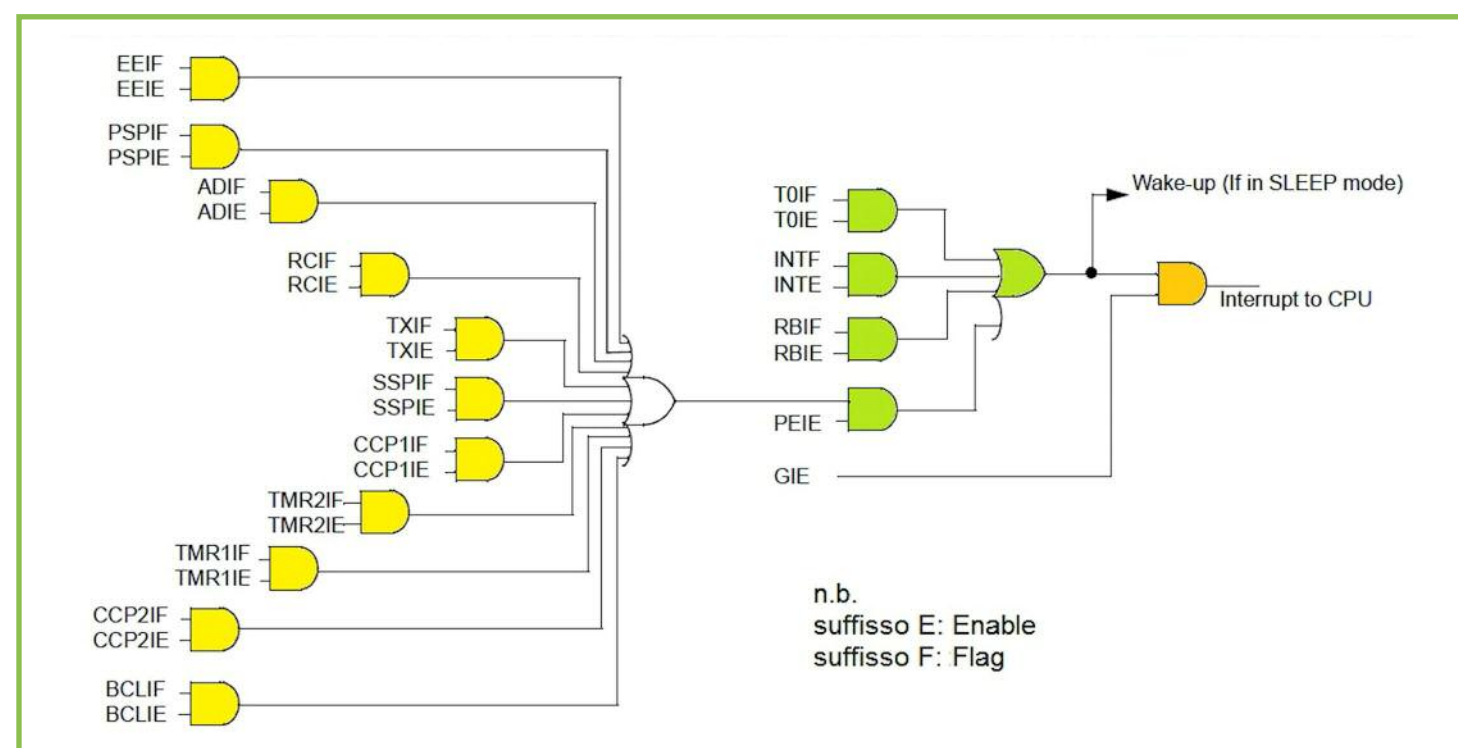



Figura 1–Schema logico-funzionale degli interrupt del PIC 16F877. Il suffisso E indica il bit di enable dell'interrupt mentre il suffisso F (Flag) indica lo stato dell'interrupt [1]. Si analizzi attentamente il contenuto della Tabella 2 per maggiori dettagli.

sata su una logica di interrogazione a controllo di programma.
A questo argomento verranno dedicate ben due lezioni di questo corso essendo fondamentale per chiunque voglia imparare a scrivere codice per controllori di media complessità ben strutturato, efficiente da un punto di vista funzionale e cosa non meno importante, garantendone in modo corretto modularità e manutenibilità.

GESTIONE DI EVENTI ASINCRONI: IL POLLING E GLI INTERRUPT

Affinché una CPU possa dialogare correttamente con dispositivi di I/O deve sincronizzarsi opportunamente con essi. I metodi per realizzare ciò sono fondamentalmente due: il polling e gli interrupt.
La tecnica del polling prevede che il processore controlli periodicamente, secondo una cadenza prefissata lo stato della periferica. In pratica si instaura un cosiddetto polling loop finalizzato all'intercettazione dell'evento. Il vantaggio di una tale gestione sta nella semplicità logica del controllo e nel fatto che non sono necessarie particolari complicazioni hardware nell'architettura della CPU; lo svantaggio sta invece in una gestione poco efficiente del tempo e in un appesantimento del polling loop, soprattutto nel caso in cui i dispositivi da controllare sono molteplici. Il problema si complica ulteriormente nel caso in cui l'evento da intercettare abbia una durata molto breve e per sua natura necessiti di essere servito nel preciso istante in cui si verifica pena l'inefficacia del servizio di risposta o addirittura la perdita vera e propria dell'evento.
La tecnica degli interrupt è da questo punto di vista completamente differente. In

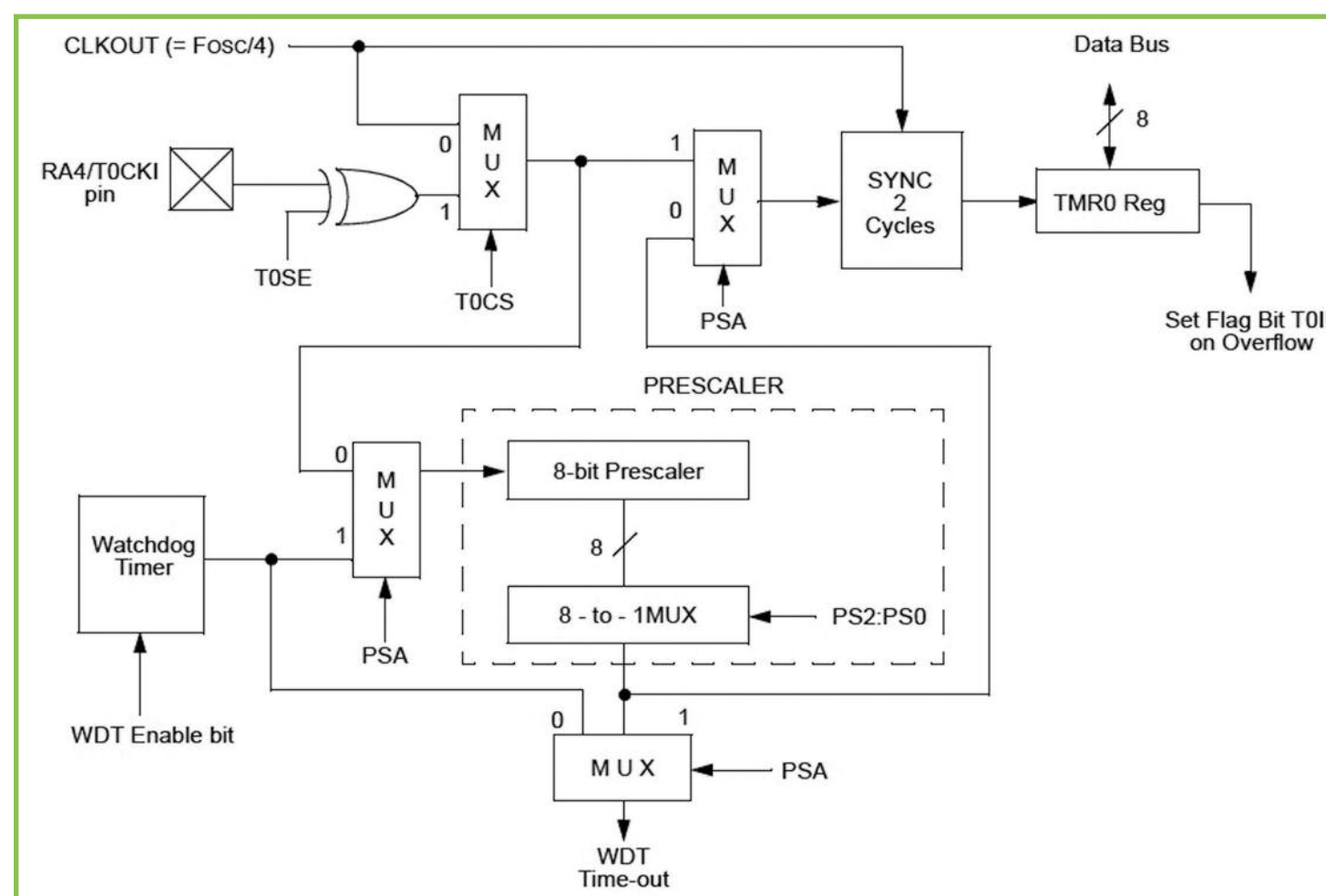



Figura 2-Schema a blocchi dell'architettura hardware interna del Timer0 del PIC16F87X. T0CS, T0SE, PSA, PS2:PS0 sono i bit del campo 5:0 del registro OPTION_REG. Si rimanda al par. 2.2.2.2- OPTION_REG Register di [1] per ulteriori dettagli.

questo caso, la CPU è dotata di ingressi di interrupt; appena l'interrupt giunge alla CPU conclude l'esecuzione dell'istruzione in corso, salva il suo stato nello stack e provvede immediatamente a servire l'interrupt; solo alla fine di tale attività ritorna ad eseguire il flusso di programma precedentemente interrotto dal punto in cui era stato interrotto. Evidentemente, la tecnica dell'interrupt consente uno sfruttamento migliore e più efficiente della CPU e allo stesso tempo una risposta più rapida alla periferica che ha generato l'interrupt. La gestione a interrupt degli eventi può evidentemente essere del tutto asincrona ma necessita allo scopo di hardware specifico. Generalmente, la maggior parte degli interrupt possono essere mascherabili cioè disabilitabili e più interrupt possono essere distinti in quanto vettorizzati. In ogni caso, a ogni interrupt corrisponde l'esecuzione di una routine finalizzata a servire l'interrupt stesso. L'argomento riguarda indistintamente processori e controllori e si complica ovviamente se si introduce la necessità di gestire una scala di priorità tra diversi interrupt. Fortunatamente, da questo punto di vista, nel caso dei controllori, il numero, la tipologia e la gestione degli interrupt è generalmente un po' meno complessa di quanto non accada nei moderni calcolatori.

GLI INTERRUPT NEI CONTROLLORI

Sebbene il processore di un computer sia generalmente ben più complesso di un controllore, il concetto di interrupt e molti degli aspetti che ne regolano la gestione, sono applicabili tanto agli uni quanto agli altri.
Fino alla scorsa puntata del corso abbiamo visto un flusso di programma come qualcosa di strettamente sequenziale e dall'evoluzione perfettamente prevedibile anche in presenza di chiamate a funzioni o procedure. Il flusso di programma inoltre ci è fin qui apparso come interamente contenuto all'interno di un loop infinito che viene cioè ripetuto indefinitamente (ciclo while principale degli esempi).
Un processo di controllo implementato rigorosamente secondo questo modello comporta dei chiari limiti, come abbiamo già in parte compreso, poiché implica una gestione a polling degli eventi. Se, per esempio, alla pressione di un pulsante deve corrispondere immediatamente una determinata operazione, questa potrebbe non essere eseguita o non essere eseguita in tempo utile se in quel momento il flusso di programma è impegnato dall'esecuzione di operazioni che richiedono un certo tempo. Se si vuole che il programma risponda immediatamente al verificarsi dell'evento, eseguendo immediatamente determinate operazioni per poi tornare al normale funzionamento, è necessario che l'architettura del processore o nel nostro caso del controllore integri quindi meccanismi di interrupt.
Evidentemente, un programma che prevede l'uso di interruzioni non si compone solo di un loop principale, all'interno del quale possono eventualmente esserci anche chiamate a funzioni o procedure, ma anche di routine apposite dette appunto di interrupt.

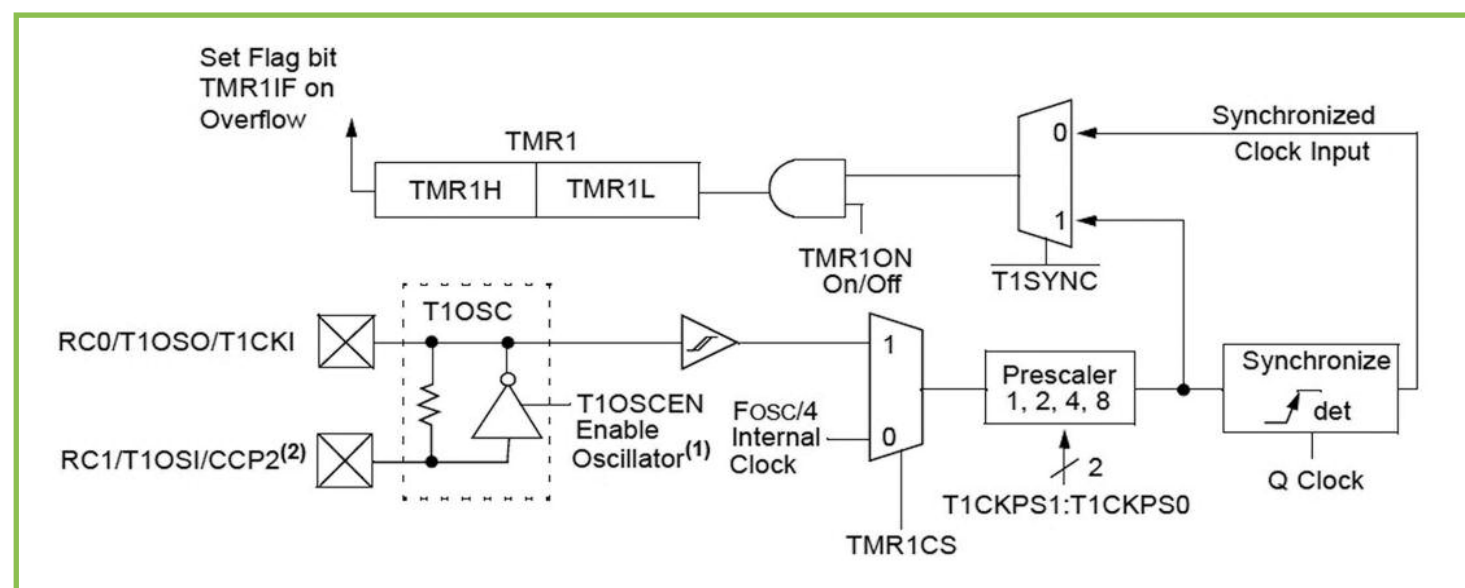



Figura 3-Schema a blocchi funzionale del Timer1 [1]

Interrupt significa quindi e in pratica la possibilità di gestirne prontamente l'imprevedibilità stessa di un evento. L'interrupt viene servito e quindi risolto grazie al salto a una routine apposita di gestione e risoluzione, tuttavia esso non equivale ad una semplice chiamata di una funzione o di procedura contrariamente a quanto una prima sommaria descrizione potrebbe portare a pensare. Infatti, mentre una procedura o una funzione può essere chiamata da punti precisi del flusso di programma, il salto alla routine di risoluzione di un interrupt (ISR-Interrupt Service Routine) può avvenire al verificarsi dell'evento, qualunque sia il punto in cui si trovi localizzata l'esecuzione del codice in quel momento.
Questo è il motivo per cui un controllore dotato di interrupt può gestire in maniera efficiente e asincrona eventi esterni quali allarmi derivanti da contatti, pulsanti, sensoristiche di vario genere e comunicazione di dati. L'approccio a interrupt consente quindi di sincronizzare in maniera efficace il flusso di programma con eventi esterni anche per loro natura imprevedibili.

Interrupt possono scaturire anche dalla ricezione di un carattere su una comunicazione UART, dalla avvenuta conversione AD eseguita attraverso l'ADC intero del controllore, dall'overflow di risorse interne quali i timer. Nel caso della comunicazione UART, la corrispondente ISR è interessata solo al ricevimento del carattere mentre in qualunque altro frangente di tempo il flusso di programma non viene affatto interessato, analogamente nel caso dei timer è possibile eseguire conteggi temporali estremamente precisi del tutto indipendenti dal flusso di programma cioè dall'impiego all'interno di questo di istruzioni di ritardo (delay). Evidentemente, l'impiego di tecniche di interrupt nella gestione degli eventi modifica l'aspetto funzionale del codice che dal punto di vista esecutivo, in determinati istanti, quelli in cui si verifica l'interrupt appunto, devia dall'esecuzione sequenziale classica.
E' interessante osservare come l'attivazione di una ISR al verificarsi di un interrupt viene sempre preceduta dal salvataggio dello stato del processore o del controllore in quell'istante. In pratica si tratta di salvare nello stack della macchina il Program Counter (indirizzo dell'istruzione che si stava per eseguire al momento dell'evento di interrupt) ed altri registri di stato che l'ISR potrebbe modificare durante la sua attività (Figura 5). In ogni caso, appare chiaro come la possibilità di impiegare tecniche di interrupt non sia legata solo ed esclusivamente ad aspetti firmware (o software nel caso dei calcolatori) ma anche ad aspetti architetturali propri dell'hardware. Non è possibile scrivere un firmware che consenta la gestione di un processo a interrupt se il controllore non è dotato delle risorse necessarie. Il fatto di disporre di risorse di interrupt non significa comunque che ilo sviluppatore del firmware non possa scegliere di gestire determinati eventi in maniera puramente sequenziale sfruttando la logica del polling.
Ovviamente, la scelta dipende spesso dalla complessità del processo e da conside-

Figura 4-Schema a Blocchi funzionali del Timer2 del PIC 16F877 [1]

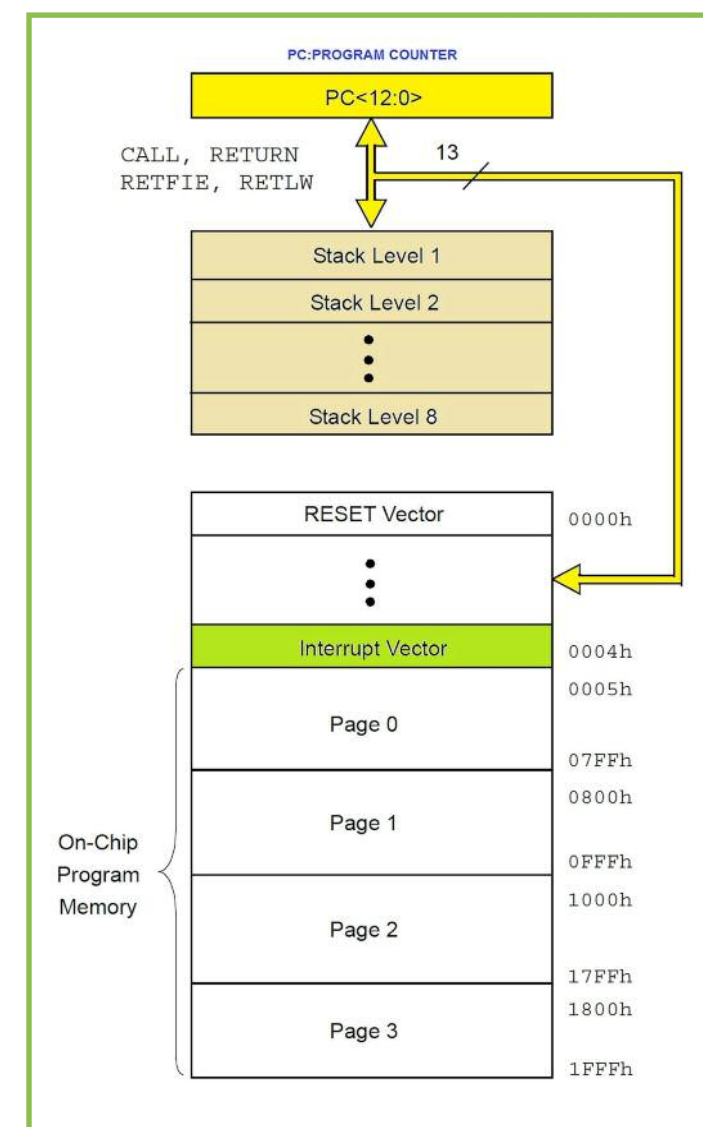



Figura 5– Mappa della memoria di programma e dello stack del PIC 16F877. Si noti in particolare il reset vector e l'interrupt vector [1]

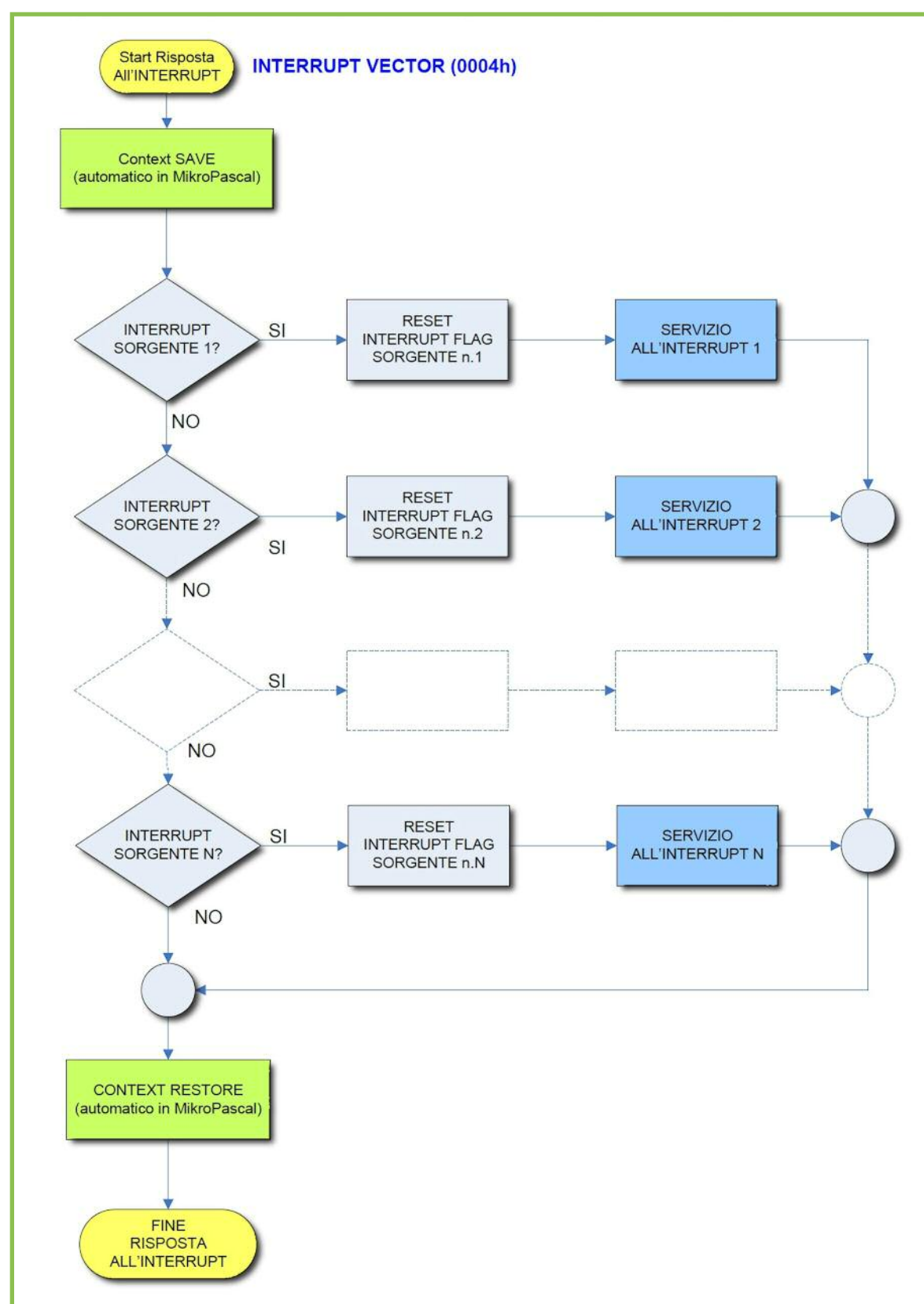



Figura 6-Diagramma di flusso esprimente la tipica gestione di un interrupt per un controllore PIC programmato in linguaggio ad alto livello

razioni di opportunità che variano caso per caso in funzione della specifica applicazione. In linea del tutto generale tuttavia, più il flusso diviene complesso è più il codice diventa efficiente e ordinato se si sfruttano adeguatamente, dove possibile, risorse di interrupt.
Specie in questi casi infatti, un corretto uso degli interrupt semplifica il codice per due motivi fondamentali:

- ne aumenta la modularità;
- non vincola la stesura del codice a strutture e temporizzazioni poco efficienti e rigide sia da un punto di vista funzionale che da un punto di vista di manutenibilità.

E' importante sottolineare che l'impiego di codice che fa uso di interrupt può essere sottoposto a debug in maniera meno semplice proprio perché gli interrupt rompono la classica concezione di esecuzione sequenziale. Quindi l'impiego degli interrupt richiede allo sviluppatore del firmware del micro un salto mentale nella sua capacità di vedere e strutturare il flusso di programma.

GLI INTERRUPT DEL PIC

Un interrupt è quindi un evento che causa, a prescindere dal punto in cui il flusso di programma si trovi in quel momento, l'esecuzione di una subroutine residente in uno speciale porzione della memoria di programma.
Gli interrupt sono il meccanismo più rapido e quindi efficiente per costringere la MCU a rivolgere prontamente l'attenzione a un problema della massima urgenza ma vanno utilizzati con estrema attenzione. I controllori come il PIC16F877 sono grazie a queste risorse molto di più che delle piccole CPU con della RAM e della ROM integrate al loro interno. Al fine di entrare nello specifico degli interrupt dei controllori PIC, analizzeremo da questo momento in poi, gli interrupt dei controllore PIC16F87X [1] e del PIC 16F877 in particolare per due principali motivi:

- il PIC16F877 è quello che abbiamo fino a questo momento utilizzato e che continueremo prevalentemente a utilizzare nel proseguo del corso come MCU su cui sviluppare gli esempi di progetto;
- quanto sarà esposto a riguardo vale comunque nelle linee generali anche per altri modelli di PIC.

Interrupt del PIC 16F87X

Il PIC16F877 ha complessivamente 14 interrupt a ciascuno dei quali è associato un "Interrupt Enable Bit" che va impostato ad 1 per abilitare l'interrupt oppure a 0 per la disabilitarlo ed un "Interrupt Flag" che assume valore 1 quando l'interrupt si attiva. E' importante sottolineare la differenza tra interrupt enable bit che serve per abilitare l'interrupt ed interrupt flag che rappresenta a tutti gli effetti lo stato dell'interrupt.
Tra gli interrupt ne figurano due che possiamo definire di carattere globale: il "Global Interrupt Enable Bit" (GIE) e il "Peripheral Interrupt Enable Bit" (PEIE). Il primo deve essere impostato prima dell'abilitazione di qualunque altro interrupt. Il secondo invece deve essere impostato a 1 prima che venga abilitato qualunque altro tipo di interrupt di periferica.
Quando si verifica un interrupt, parte una chiamata ad un Interrupt Service Routine or "ISR" la quale altro non è che una subrotine residente all'indirizzo della memoria di programma 0x0004 (interrupt vector, Figura 6).
Prima del ritorno dall'ISR l'utente deve eseguire il reset dell' "Interrupt Flag" che era stato portato ad 1 nel momento in cui si era verificato l'interrupt stesso. Questa azione è di estrema importanza poiché se non eseguita, l'interrupt hardware causerebbe automaticamente una nuova e immediata interruzione appena viene eseguita l'istruzione di ritorno dall'interrupt.
E' anche opportuno fare attenzione al fatto che involontariamente potrebbe determinarsi un evento di interrupt quando viene impostato ad 1 il bit "Interrupt Enable Bit" cosa che accade se l'"Interrupt Flag" si trova già a livello alto.
In maniera esplicita, in caso di programmazione assembly, e in buona parte nascosta al programmatore, nel caso di im-

piego di compilatore ad alto livello, in occasione di un interrupt vengono eseguite le seguenti azioni:

- salvataggio in dei registri di stato della MCU;
- esame dello stato degli "Interrupt Flags" al fine di determinare quale interrupt si è verificato;
- esecuzione dell'ISR.

Prima di effettuare il ritorno dalla ISR vengono eseguite invece le seguenti azioni:

- reset dell'interrupt flag interessato;
- se vi sono altri "Interrupt Flags" a valore 1 vengono serviti anche essi;
- ripristino dei valori de registri di stato;
- ritorno dalla ISR;
- si ripete quanto esposto per ciascun interrupt fino a che non vi sono ulteriori interrupt flag attivi.

Il diagramma di flusso di Figura 6 esprime efficacemente questo concetto.

Occorre a questo punto, con riferimento ai controllori PIC16F87X soffermare maggiormente l'attenzione su due aspetti:

-come si scatenano o meglio quali eventi possono scatenare un interrupt;

-come si abilitano gli interrupt e come è possibile gestirli.

I controllori PIC, come altri controllori, integrano numerosi interrupt hardware sia esterni che interni. Con i termini esterno ed interno si indica la provenienza dell'interrupt. Per esempio, il cambio di stato di una porta utilizzata come ingresso è un interrupt esterno mentre l'overflow del registro timer è un interrupt interno. Analogamente, il completamento di una conversione AD può essere considerato un interrupt interno mentre l'arrivo di un dato su una comunicazione seriale può essere considerato un interrupt esterno.

Lo stato di uno specifico interrupt è dato dal valore assunto da uno specifico bit di un opportuno registro che deve essere gestito opportunamente. Tanto per cominciare, è necessario da parte del programmatore gestire l'abilitazione e la disabilitazione dell'interrupt. Riferendoci alla famiglia di controllori PIC16F, cui appartengono i PIC 16F87X, il vettore di interrupt è localizzato all'indirizzo 0004H della memoria (Figura 6).

Lo schema di Figura 7 illustra la logica con cui vengono serviti gli interrupt nel PIC. Quando si scatena un interrupt la CPU ne riceve informazione e attiva una routine di controllo tesa ad analizzare il vettore di interrupt per scoprire quale interrupt è stato scatenato al fine di servirlo con un opportuno ISR (Interrupt Service Routine).

In una programmazione a basso livello cioè in linguaggio assembly la gestione dell'interrupt comporta operazioni di context save e context restore a carico dello sviluppatore del firmware, rispettivamente all'inizio e alla fine della gestione dell'interrupt. In sostanza, significa rispettivamente salvare all'inizio dell'interrupt e ripristinare alla fine della sua gestione i valori dei registri che potenzialmente potrebbero essere modificati dall'ISR. Questa complicazione viene meno, nel senso che è nascosta al programmatore, con l'impiego di compilatori ad alto livello. In questo caso infatti dette operazioni, che possiamo definire più in generale di context switch, vengono gestite in automatico dal compilatore rimanendo quindi nascoste alo sviluppatore.

I 14 interrupt del PIC16F877 sono rispettivamente:

- cambiamento di stato sulla linea RB0 (external interrupt, INT Pin Interrupt);
- fine conteggio del registro TMR0 (TMR0 Overflow Interrupt);
- cambiamento di stato delle linee che vanno da RB4 a RB7 (PORTB Change Interrupt);
- cambiamento di stato del modulo Comparatore (Comparator Change Interrupt);
- interrupt sulla porta parallela (Parallel Slave Port Interrupt);
- interrupt sulla porta seriale (USART Interrupt);
- receive interrupt;
- transmit interrupt;
- fine di una conversione A/D (A/D Conversion Complete Interrupt);
- interrupt sul modulo LCD (LCD Interrupt);
- fine della scrittura su una locazione EEPROM (Data EEPROM Write Complete Interrupt);
- fine del conteggio del timer TMR1 (Timer1 Overflow Interrupt);
- interrupt sul modulo Capture/Compare (CCP Interrupt);

```
INTCON.GIE := 0;   // Azzeramento del Global Interrupt Bit (GIE)

INTCON.B0 := 0;    // Azzera il bit 0 del registro INTCON
ADCON0.5 := 1;     // Porta ad 1 il valore del bit 5 del registro ADCON0
i := 5;
STATUS.(i+1) := 1;// Imposta ad 1 il bit 6 del registro di stato
```

Figura 7-Esempio di referenziazione dei bit dei registri di interrupt [2],[3]

BIT	Acronimo	Significato
bit 7	**GIE**: Global Interrupt Enable bit	1 = Abilitazione generale degli interrupt 0 = Disabilitazione generale degli interrupt
bit 6	**PEIE**: Peripheral Interrupt Enable bit	1 = Abilitazione degli interrupt di periferica 0 = Disabilitazione degli interrupt di periferica
bit 5	**T0IE**: **TMR0** Overflow Interrupt Enable bit	1 = Abilitazione dell'interrupt TMR0 0 = Disabilitazione dell'interrupt TMR0
bit 4	**INTE**: RB0/INT External Interrupt Enable bit	1 = Abilitazione dell'interrupt RB0/INT 0 = Disabilitazione dell'interrupt RB0/INT
bit 3	**RBIE**: **RB** Port Change Interrupt Enable bit	1 = Abilitazione dell'interrupt RB port change 0 = Disabilitazione dell'interrupt RB port change
bit 2	**T0IF**: **TMR0** Overflow Interrupt Flag bit	1 = TMR0 in overflow (da azzerare in firmware) 0 = TMR0 non in overflow
bit 1	**INTF**: RB0/INT External Interrupt Flag bit	1 = interrupt RB0/INT avvenuto (da azzerare in firmware) 0 = interrupt RB0/INT non avvenuto
bit 0	**RBIF**: **RB** Port Change Interrupt Flag bit	1 = Almeno uno dei pin RB7:RB4 ha cambiato stato 0 = Nessuno dei pin RB7:RB4 ha cambiato stato

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Tabella 1–Significato dei bit dell'INTCON Register dei PIC 16F87X[1]

BIT	Registro	Significato
EEIF	PIR2	EEIF: EEPROM Write Operation Interrupt Flag bit 1 = scrittura completata (da resettare via firmware); 0 = non completata o non ancora avviata
EEIE	PIE2	EEIE: EEPROM Write Operation Interrupt Enable bit 1 = EEPROM write interrupt abilitato; 0 = EEPROM write interrupt disabilitato
PSPIF	PIR 1	PSPIF(1): Parallel Slave Port Read/Write Interrupt Flag bit 1 = Lettura o scrittura in corso (da azzerare in firmware); 0 = Nessuna lettura o scrittura
PSPIE	PIE1	PSPIE: Parallel Slave Port Read/Write Interrupt Enable bit 1 = interrupt PSP di scrittura/lettura abilitato; 0 = interrupt PSP di scrittura/lettura disabilitato
ADIF	PIR 1	ADIF: A/D Converter Interrupt Flag bit 1 = conversione A/D completata; 0 = conversione A/D non completata
ADIE	PIE1	ADIE: A/D Converter Interrupt Enable bit 1 = interrupt della conversione A/D abilitato; 0 = interrupt della conversione A/D disabilitato
RCIF	PIR1	RCIF: USART Receive Interrupt Flag bit 1 = buffer USART pieno; 0 = buffer USART vuoto
RCIE	PIE1	RCIE: USART Receive Interrupt Enable bit 1 = Interrupt ricezione USART abilitato; 0 = Interrupt ricezione USART disabilitato;
TXIF	PIR1	TXIF: USART Transmit Interrupt Flag bit 1 = buffer USART TX vuoto; 0 = buffer USART TX pieno;
TXIE	PIE1	TXIE: USART Transmit Interrupt Enable bit 1 = interrupt USART TX abilitato; 0 = interrupt USART TX disabilitato
SSPIF	PIR1	SSPIF: Synchronous Serial Port (SSP) Interrupt Flag 1 = interrupt SSP verificatosi (da azzerare in firmware prima dell'uscita dalla routine di servizio dell'interrupt). Questo interrupt riguarda le trasmissioni SPI, I2C Slave, I2C Master
SSPIE	PIE1	SSPIE: Synchronous Serial Port Interrupt Enable bit 1 = interrupt SSP abilitato; 0 = interrupt SSP disabilitato
CCP1IF	PIR1	CCP1IF: CCP1 Interrupt Flag bit Capture mode: 1 = TMR1 register capture verificatosi (da azzerare in firmware) 0 = TMR1 register capture non verificatosi Compare mode: 1 = TMR1 register compare match verificatosi (da azzerare in firmware) 0 = TMR1 register compare match non verificatosi Non utilizzano in PWM mode
CCP1IE	PIE1	CCP1IE: CCP1 Interrupt Enable bit 1 = Interrupt CCP1 abilitato 0 = Interrupt CCP1 disabilitato
TMR2IF	PIR1	TMR2IF: TMR2 to PR2 Match Interrupt Flag bit 1 = TMR2 to PR2 match verificatosi (da azzerare in frmware); 0 = TMR2 to PR2 match non verificatosi
TMR2IE	PIE1	TMR2IE: TMR2 to PR2 Match Interrupt Enable bit 1 = interrupt TMR2 to PR2 match abilitato: 0 = interrupt TMR2 to PR2 match non abilitato:
		1 = interrupt TMR1 overflow abilitato; 0 = interrupt TMR1 overflow non abilitato
CCP2IF	PIR2	CCP2IF: CCP2 Interrupt Flag bit Capture mode: 1 = TMR1 register capture verificatosi (da azzerare in firmware) 0 = TMR1 register capture non verificatosi Compare mode: 1 = TMR1 register compare match verificatosi (da azzerare in firmware) 0 = TMR1 register compare match non verificatosi Non utilizzato in PWM
CCP2IE	PIE2	CCP2IE: CCP2 Interrupt Enable bit 1 = Enables the CCP2 interrupt; 0 = Disables the CCP2 interrupt
BCLIF	PIR2	BCLIF: Bus Collision Interrupt Flag bit 1 = SSP bus collision in I2C Master mode; 0 = Nessuna collisione
BCLIE	PIE2	BCLIE: Bus Collision Interrupt Enable 1 = Bus Collision Interrupt abilitato; 0 = Bus Collision Interrupt non abilitato
Significato dei registri		
PIR2		registro contenente i bit flag per gli interrupt CCP2, SSP, di scrittura su EEPROM e relativi alle operazioni dei comparatori
PIE2		registro contenente i bit di enable per gli interrupt CCP2, SSP, scrittura su EEPROM e relativi alle operazioni dei comparatori
PIR 1		registro contenente i bit flag degli interrupt delle periferiche
PIE1		registro contenente i bit di enable per gli interrupt di periferica
PIR2		Registro contenente i bit flag di interrupt per CCP2, SSP, di scrittura su EEPROM e ralativi alle operazioni dei comparatori.

Tabella 2 [1]-Significato dei bit di registro che interessano gli interrupt presenti nello schema di Figura 1 ma non appartenenti al registro INTCON e quindi non riportati all'interno della Tabella 1

- interrupt sulla porta seriale sincrona (SSP Interrupt).

Abbiamo detto che al verificarsi dell'interrupt, il PIC interrompe l'esecuzione del programma in corso, memorizza automaticamente nello stack il valore corrente del program counter e del registro di stato e salta all'istruzione presente nella locazione di memoria 0004h (Interrupt Vector; Figura 5). Il motivo di saltare proprio questo indirizzo sta nel fatto che in questo punto della memoria è localizzata la subroutine di gestione degli interrupt. Potendo utilizzare più interrupt, all'interno della subroutine si verifica quale, tra gli eventi abilitati, ha generato l'interrupt analizzando gli interrupt flag in modo da eseguire la parte di codice relativa (Figura 7). Infatti, qualunque interrupt genera una chiamata alla locazione 0004h per cui è necessario analizzare il registro INTCON che contiene i flag di interrupt per rilevarne lo stato. Facciamo alcuni esempi al fine di comprendere meglio il concetto.

Quando viene generato un interrupt il PIC disabilita automaticamente il bit GIE del registro INTCON in modo che risultino disabilitati tutti gli interrupt mentre è già in esecuzione un interrupt handler al fine di impedire che possano verificarsi situazioni ricorsive; i bit di flag vengono quindi riazzerati prima di riabilitare l'interrupt globale. In Tabella 1 si riporta, in dettaglio, l'insieme dei bit componenti il registro INTCON ed il relativo significato.

In Tabella 2 riportiamo invece il significato dei bit di registro che interessano gli interrupt presenti nello schema di Figura 1 ma non appartenenti al registro INTCON e quindi non riportati in Tabella 1.

I TIMER

E' piuttosto intuitivo immaginare il timer come qualcosa che consente di contare e quindi di temporizzare. I controllori PIC integrano, nella maggior parte dei casi, timer cioè registri a 8 o 16 bit (a differenza dei PIC delle fasce più evolute come PIC24, dsPIC e PIC32 che dispongono anche di registri timer a 32 bit). Il valore di questi registri viene incrementato in funzione del tempo, scandito dalla frequenza di clock oppure da opportune sorgenti esterne.

La prima cosa che occorre comprendere del funzionamento dei timer è che il loro incremento non si verifica necessariamente ad ogni impulso di clock. La modalità di incremento può essere modificata imponendo dei cosiddetti prescaler e postscaler che fanno da divisori di frequenza al fine di ottenere la frequenza di incremento desiderata. Il numero di timer e la loro tipologia (numero di bit), come già sottolineato, dipende dal modello di controllore, e in particolare dalla famiglia di appartenenza (per esempio molti PIC della famiglia PIC12 dispongono di Timer0 e Timer 1 mentre quelli della famiglia 16F87x dispongono di Timer0, Timer1 e Timer2). In ogni caso, la logica del loro funzionamento si basa su clock interno o esterno, prescaler e postscaler.

Dalla flessibilità che ne deriva scaturisce la loro utilità nella gestione di temporizzazioni e indirettamente per ottenere funzioni PWM (Pulse Width Modulation) e il CCP (Capture/Compare/PWM) o per scatenare un evento conseguente all'overflow del timer stesso. Di seguito analizziamo più in dettaglio l'architettura interna dei timer del PIC 16F87X (Timer0, Timer1 e Timer2).

Timer0 dei PIC 16F87X

Il Timer0 (registro TMR0, Figura 2) è un timer 8 bit che condivide il prescaler con il **W**atch**D**og **T**imer (WDT). Questo significa che è necessario assegnare in maniera mutuamente esclusiva il prescaler all'una o all'altra risorsa. La condivisione della risorse tra i due circuiti, secondo una logica di multiplexing, è infatti ricorrente nelle architetture dei controllori poiché contribuisce a integrare più funzionalità mantenendo comunque snello l'hardware. Il registro TMR0 incrementa il suo valore sotto l'influenza della sorgente di oscillazione interna Fosc/4 oppure sotto l'influenza di un'oscillazione esterna (ingresso T0CKI che per il PIC16F877 corrisponde con il pin RA4).

Il senale di clock così definito e intercettato può essere modificato ulteriormente internamente al controllore mediante il circuito di prescaler che ne effettua una divisione di frequenza corrispondente a un fattore moltiplicativo $1:2^n$, con n compreso tra 1 ed 8. Il fattore moltiplicativo risulta quindi compreso tra 1:2 ed 1:256. Ovviamente, l'effetto del prescaler sul timer si manifesta solo se esso è assegnato al timer, se esso è invece assegnato al WDT non ha alcun effetto sul timer che di conseguenza funziona con fattore di prescaler pari ad 1:1.

La configurazione del timer avviene agendo sul registro OPTION_REG. In pratica, assegnando il valore opportuno a questo registro, si esegue:

- la scelta della sorgente di clock (interna/esterna);
- la scelta della risorsa cui assegnare il prescaler (Timer/WDT);
- l'impostazione del valore di prescaler (fattore di divisione).

Il prescaler viene impostato agendo sul bit PSA del registro OPTION_REG (Figura 2). In pratica, il bit PSA viene posto a zero (si noti il MUX su PSA in Figura 2) e il fattore di prescaler resta così definito attraverso i bit (PS2, PS1, PS0) secondo la seguente assegnazione:

(0,0,0) à 1:2	(1,0,0) à 1:32
(0,0,1) à 1:4	(1,0,1) à 1:64
(0,1,0) à 1:8	(1,1,0) à 1:128
(0,1,1) à 1:16	(1,1,1) à 1:256

Per impostare Timer0 come timer, è sufficiente azzerare il bit TOCS del registro OPTION_REG (OPTION_REG<5>). Con questa impostazione infatti, come mostra chiaramente lo schema di Figura 2, il registro Timer0 si incrementa sotto il comando del clock di sistema e più precisamente a ogni ciclo di istruzione cioè con frequenza Fosc/4 (senza prescaler; n.b ciascuna istruzione del set di istruzioni di basso livello impiega quatto cicli di clock per essere eseguita). Per impostare invece Timer0 come contatore è necessario portare il bit TOCS (OPTION_REG<5>) a valore 1. In queste condizioni, ogni fronte del segnale presente su RA4/T0CKI determina un incremento del registro. Nella modalità da contatore è inoltre possibile impostare il bit T0SE del registro OPTION_REG a zero per effettuare il conteggio sul fronte di discesa e a 1 per effettuare il conteggio sul fronte di salita. Anche il fronte del segnale di comando sul pin RA4 (TOCKI) può essere fissato attraverso il bit T0SE (Timer0 Source Edge, OPTION_REG<4>). Se il bit T0SE è impostato a 0 vale come attivo il fronte di salita mentre se è impostato ad 1 vale come attivo il fronte di discesa. Per maggiori dettagli in merito si rimanda ai paragrafi di [1]:

- par.5.2 "Using Timer0 with an External Clock;
- par. 5.3 "Prescaler" details the operation of the prescaler.

Ovviamente, nel funzionamento da contatore, essendo "T0CKI" coincidente con il pin RA4 è necessario definire il pin come ingresso attraverso l'impostazione del registro TRISA.

Timer1

Con riferimento al solito PIC 16F87X, Timer 1 (registro TMR1-Figura 3) è, diversamente da Timer 0, un timer/contatore a 16 bit e presenta quattro soli fattori di prescaler (1:1, 1:2, 1:4 e 1:8). Diversamente da quanto accade con il Timer0 inoltre, il prescaler è dedicato cioè non è condiviso con altre risorse o funzionalità. La sorgente interna è la soilta Fosc/4 mentre quella esterna è T1CKI coincidente con il pin RC0.

Timer2

E' un timer ad 8-bit dotato di prescaler e postscaler (Figura 4). Il fattore di prescaler sulla sorgente di clock (FOSC/4) può assumere valore 1:1, 1:4 e 1:16, selezionabili attraverso l'impostazione dei bit T2CKPS1:T2CKPS0 (T2CON<1:0>). Analogamente, il fattore di postscaler può assumere i seguenti valori in base all'impostazione de 4 bit TOUTPS3:TOUTPS0:

(0,0,0,0) à 1:1	(1,0,0,0) à 1:9
(0,0,0,1) à 1:2	(1,0,0,1) à 1:10
(0,0,1,0) à 1:3	(1,0,1,0) à 1:11
(0,0,1,1) à 1:4	(1,0,1,1) à 1:12
(0,1,0,0) à 1:5	(1,1,0,0) à 1:13
(0,1,0,1) à 1:6	(1,1,0,1) à 1:14
(0,1,1,0) à 1:7	(1,1,1,0) à 1:15
(0,1,1,1) à 1:8	(1,1,1,1) à 1:16

Diversamente dai precedenti due timer, Timer2 presenta un registro a 8 bit di impostazione del periodo (PR2). Grazie ad esso Timer2 può incrementare il suo valore da 0 fino al valore impostato in PR2. Raggiunto questo valore, il conteggio riparte (quindi anticipatamente) da 0. PR2 è per questo un registro scrivibile che viene automaticamente inizializzato al valore massimo FFh ad ogni reset. TMR2IF (PIR1<1>) costituisce l'interrupt. Ponendo a zero il bit di controllo TMR2ON (T2CON<2>) è possibile arrestare il conteggio.

EVOLUZIONE DEL TIMER

Supponendo che il timer sia pilotato dalla sorgente interna e di far riferimento al Timer0, il registro TMR0 incrementa il suo valore con una frequenza pari ad Fosc/4, ulteriormente ridotta di un fattore di divisione pari al valore di prescaler. Se per esempio abbiamo una sezione di oscillazione di frequenza 8MHz come quella utilizzata negli esempi delle precedenti lezioni, e un prescaler pari ad 1:8, avremo l'incremento del timer con frequenza ((8000000/4)/8)Hz=250 kHz che corrisponde ad un incremento ogni 4 us. Di conseguenza, il registro TMR0 essendo esteso 8 bit, subirà un overflow ogni 256 x 4us = 1024us = 1,024 ms. Un ulteriore grado di libertà nella gestione del timer è offerto dalla possibilità di precaricare un valore all'interno del registro stesso (è il caso del Timer 2 descritto nel precedente paragrafo) in modo da anticipare l'overflow.

INTERRUPT ON-CHANGE

L' Interrupt On Change (IOC) è un meccanismo di interrupt innescato dalle transi-

zioni di stato logico su pin di I/O. Si tratta di un meccanismo di interrupt estremamente utile in una infinità di situazioni operative che vanno dal semplice rilevamento della pressione di un pulsante alla transizione di contatti o stati logici provenienti da sensori come fotocellule o dispositivi dei più disparati come microswitch e fotoaccoppiatori. Sulla serie di PIC PIC16F87X tali interrupt sono associati ai pin PORTB<7:4>, utilizzati ovviamente come ingressi.

Nella prossima puntata illustreremo alcuni esempi di impiego degli interrupt on-change, possiamo tuttavia anticipare fin da ora quelle che sono le fasi essenziali del loro utilizzo:

- corretta configurazione del registro IOC (impostazione a 1 dei bit delle porte interessate);
- abilitazione (impostazione a 1) del bit GPIE e GIE del registro INTCON;
- risoluzione dell'interrupt attraverso l'esecuzione del necessario blocco di codice.

E' importante sottolineare come l'interrupt on-change non fornisca automaticamente risposta sul segno della transizione ma solo sul fatto che la transizione sia avvenuta. La valutazione del tipo di transizione (Vdd→GND o viceversa) deve essere fatta direttamente in firmware dalla routine che risolve l'interrupt.

EXTERNAL INTERRUPT RBO/INT

Un interrupt simile all'interrupt-on change può essere attivato sul pin RB0. Si tratta di un interrupt che può essere rilevato come transizione in salita sul pin RB0 e che può anche essere utilizzato per portare il controllore fuori da uno stato di sleep.

Organizzazione della memoria e Interrupt

Ci sono tre blocchi di memoria all'interno dei controllori PIC16F87X (Figura 5). La memoria di programma e la memoria dati utilizzano bus distinti che consentono accesso concorrente. Il PIC16F87X integra un program-counter a 13 bit che consente di conseguenza di indirizzare 8Kx14 words di memoria FLASH. L'indirizzo 0000h rappresenta il reset vector mentre l'indirizzo 0004h rappresenta l'interrupt vector.

Il PIC inizia l'esecuzione del firmware a partire dal cosiddetto "vettore di reset" (reset vector) cioè dall'istruzione presente nella prima locazione di memoria (000h) e prosegue con l'esecuzione delle istruzioni memorizzate nelle locazioni successive. Il program-counter mantiene traccia dell'indirizzo della prossima istruzione da eseguire. Le istruzioni di salto come quella corrispondente al loop principale modificano tale flusso esecutivo naturale per cui il flusso di programma in effetti non raggiunge praticamente mai l'ultima locazione di memoria disponibile.

Al reset corrisponde un azzeramento del program-counter e il reinizio pertanto dell'esecuzione a partire dall'indirizzo 0000h (è questo il motivo per cui tale indirizzo prende il nome di reset vector).

In qualche modo, anche un evento di interrupt è una modifica del flusso di programma. In questo caso infatti il program-counter viene forzato all'indirizzo 0004h.

MIKROPASCAL PRO FOR PIC E GLI INTERRUPT

In questo paragrafo forniamo alcune indicazioni generali sull'uso degli interrupt in codice scritto in MikroPascal PRO for PIC.

```
procedure interrupt;
begin
...
...
end;
```
(1)

```
procedure interrupt_low;
begin
...
...
end;
```
(2)

```
procedure Timer0_Interrupt(); iv 0x000008; ics ICS OFF;
begin                        // Timer0 interrupt routine
counter := counter + 1;
TMR0 := 96;
INTCON := $20;
end;
```
(3)

Figura 8-Esempi di sintassi si procedure di interrupt scritte in MikroPascal PRO for PIC [2],[3]

In MikroPascal (ma la stessa cosa accade in MikroC e MikroBasic) la gestione degli interrupt è più semplice che nella programmazione a basso livello tipica dell'assembly, se non altro perché le operazioni di context save e context restore si sviluppano automaticamente a carico del compilatore e rimanendo pertanto nascoste allo sviluppatore del firmware. La routine di risposta all'interrupt deve invece essere scritta dal programmatore. In linea del tutto generale, il diagramma di flusso di Figura 6 esprime questo concetto.
Il compilatore capisce che una porzione di codice è una routine di risposta all'interrupt perché scritta in una procedura denominata "procedure interrupt()";
E' possibile accedere ai singoli bit dei registri in maniera estremamente semplice. Se prendiamo per esempio in esame il Global Interrupt Bit (GIE), nello scrivere codice MikroPascal Pro for PIC potremo accedere a esso direttamente attraverso un'istruzione del tipo riportata in Figura 7. Come si osserva è possibile referenziare il singolo bit anche facendo diretto riferimento alla sua posizione all'interno del registro utilizzando la solita notazione puntata e tenendo conto che l'indice 7 indica sempre il bit più significativo del registro (essendo questo ad 8 bit) e 0 quello meno significativo.
Questo secondo modo di agire presenta potenzialmente un pericolo in più: quello di dover essere certi non solo di agire sul registro corretto ma anche sul bit corretto. Utilizzando invece una notazione come la prima (INTCON.GIE := 0) è più difficile incorrere in errori poiché si sta utilizzando la notazione mnemonica non solo del registro ma anche dello specifico bit sul quale si intende operare.

PROCEDURE DI GESTIONE DEGLI INTERRUPT IN MIKROPASCAL

In mikroPascal PRO for PIC le procedure di gestione degli interrupt sono indicate con il nome "interrupt" ed "interrupt_low" pertanto queste due termini sono da considerarsi a tutti gli effetti parole riservate. Facendo, in particolare, riferimento ai PIC delle famiglie 16 e 18, la parola riservata interrupt identifica un interrupt ad alta priorità che tipicamente presenta una sintassi del tipo riportato in Figura 8-(1).
I PIC della famiglia P18 possono anche avere interrupt a bassa priorità identificati dalla parola riservata "interrupt_low" e tipicamente caratterizzati da una sintassi del tipo riportato in Figura 8-(2).
MikroPascal PRO for PIC prevede il salvataggio dei seguenti SFR all'interno dello stack (Figura 5) al momento dell'ingresso in una routine di interrupt:

- famiglia PIC12: W, STATUS, FSR, PCLATH
- famiglia PIC16: W, STATUS, FSR, PCLATH
- famiglia PIC18: FSR.

Gli stessi valori vengono ripristinati alla fine della routine di interrupt. Allo scopo di facilitare l'impiego degli interrupt nella stesura di codice MikroPascal Pro for PIC, sono state introdotte le parole chiavi "iv" ed "ics". Per esempio, è possibile trovare una procedura scritta come in Fgura 8-(3) dove:

- **iv** informa il compilatore che si tratta di un ISR (Interrupt Service Routine).
- **ics** (Interrupt Context Saving) dove ICS_OFF→ No context saving; ICS_AUTO→ context saving lasciato a carico del compilatore.

Questo consente, volendo, di utilizzare un qualunque nome identificativo per le routine ISP. Operando in questo modo, è necessario specificare l'appropriato vettore di interrupt (high priority interrupt→0x000008, low priority interrupt→0x000018) accanto alla parola riservata "iv".

Chiamate di funzioni provenienti da Interrupt

MikroPascal Pro for PIC consente la chiamata di funzioni dall'interno delle routine

```
procedure interrupt();
begin
counter := counter + 1;
TMR0 := 96;
INTCON := $20;
end;
```
(1)

```
procedure interrupt();
begin
if TestBit(INTCON, TMR0IF) = 1 then
begin
counter := counter + 1;
TMR0 := 96;
ClearBit(INTCON, TMR0IF);
// ClearBit è una inline function,
// e può essere chiamata dall'interno dell'interrupt
end
else
if TestBit(INTCON, RBIF) = 1 then
begin
counter := counter + 1;

TMR0 := 96;
ClearBit(INTCON, RBIF);
end;
end;
```
(2)

Figura 9-Esempi di procedure di interrupt [2],[3]

di interrupt. Il compilatore tiene conto dei registri utilizzati interessati dal firmware sia durante l'esecuzione di codice main che di codice di interrupt salvando opportunamente i valori degli stessi.

Esempi di procedure di interrupt

In Figura 9-(1) è riportato un esempio di routine che gestisce l'interrupt derivante dall'overflow TMR0 nell'ipotesi che non vi siano altri interrupt da gestire ai fini del controllo del processo.

Nel caso in cui sia invece necessario gestire più interrupt, la routine potrebbe presentarsi come riportato in Figura 9-(2).

Come è possibile comprendere, per utilizzare adeguatamente gli interrupt è necessario conoscerne i relativi registri ed utilizzarli opportunamente e non solo saper utilizzare il linguaggio di programmazione (in questo caso MikroPascal Pro for PIC). A tale proposito è sempre opportuno avere a portata di mano il manuale della MCU di riferimento ([1] nel nostro caso).

CONCLUSIONI E APPROFONDIMENTI

L'argomento interrupt non si esaurisce con quanto esposto in questa lezione che tuttavia può essere considerata abbastanza esaustiva da un punto di vista generale e come preparazione alle applicazioni pratiche di cui necessiteranno le prossime puntate.

Concludiamo l'attuale puntata riportando di seguito una serie di paragrafi contenuti nel manuale del controllore PIC 16F877 ([1]-PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers) che possono essere utili per approfondimenti e il cui studio può aiutare a comprendere più in profondità quanto esposto:

- par. 2.0- Memory Oganization
- par. 2.2.2.2- OPTION_REG Register
- par.2.2.2.3-INTCON Register
- par.2.2.2.4- PIE1 Register
- par.2.2.2.5- PIR1 Register
- par.2.2.2.6- PIE2 Register
- par.2.2.2.7- PIR2 Register
- par.5.0- TIMER0 Module
- par.5.1- Timer0 Interrupt
- par.5.2- Using Timer0 with an External-Clock
- par.5.3- Prescaler
- par.6.0- TIMER1 Module
- par.7.0- TIMER2 Module
- par.12.10- Interrupts
- par.12.11- Context Saving During Interrupts

Sottolineiamo che una corretta comprensione dell'architettura del controllore dal punto di vista degli interrupt integrati è estremamente utile non solo per la comprensione del suo funzionamento ma anche dal punto di vista della programmazione firmware e non solo in quella a basso livello.

Il fatto di aver preso a riferimento soprattutto il PIC16F87X non è limitativo dal punto di vista dell'approccio al problema che nelle linee generali mantiene la stessa impostazione anche con riferimento ad altri controllori o famiglie di controllori PIC di microchip.

Per approfondire

[1]. PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers (www.mikrochip.com)
[2]. Help in linea MikroPascal Pro for PIC v. 5.61
[3]. MikroPascal PRO for PIC v.5.61 demo e documentazione relativa (www.mikroe.com)

di FLAVIO CRISEO

POWER SUPPLY "STEP DOWN"

(parte seconda)

Abbiamo visto come dimensionare un alimentatore switching con un ripple molto contenuto. In questa seconda parte vediamo com'è realizzato lo stadio di protezione elettronica ed inoltre i risultati ottenuti in fase di test

Dopo aver dato particolare attenzione al dimensionamento dell'induttore e al dimensionamento dei componenti principali per il buon funzionamento dell'alimentatore, ci occupiamo della sezione di protezione elettronica. Lo scopo principale è far si che la corrente massima sia fissata a 1,2A sia un condizioni di massimo carico che in condizioni di sovraccarico/cortocircuito.

foto 1: stadio di protezione elettronica contro i sovraccarichi ed i cortocircuiti. Il doppio Op.Amp LM358 svolge la funzione di amplificatore con guadagno in tensione G=3 e di schmitt trigger con soglia d'intervento per la V_H pari a 1,6V

Sappiamo che l'integrato LM2576 è in grado di darci 3A massime, nel caso di sovraccarico sappiamo che l'integrato non sarà posto in condizioni gravose proprio perché la corrente massima sarà fissata a 1,2A.

DIMENSIONAMENTO DELLA PROTEZIONE ELETTRONICA

Lo schema elettrico relativo allo stadio di protezione elettronica è visibile in **figura 1.** Sono presenti un integrato stabilizzatore di tensione in package TO-92 78L06 che garantisce una tensione stabile al circuito di protezione al variare della tensione in ingresso, un amplificatore dato da un Op.Amp. connesso in configurazione non invertente ed uno schmitt trigger. In uscita abbiamo tre BJT aventi il compito di segnalare lo stato di funzionamento dell'alimentatore indicando se è in funzione la protezione elettronica o meno.

Il trimmer TR_1 premette d'ottenere la giusta tensione in ingresso al trigger (la indi-

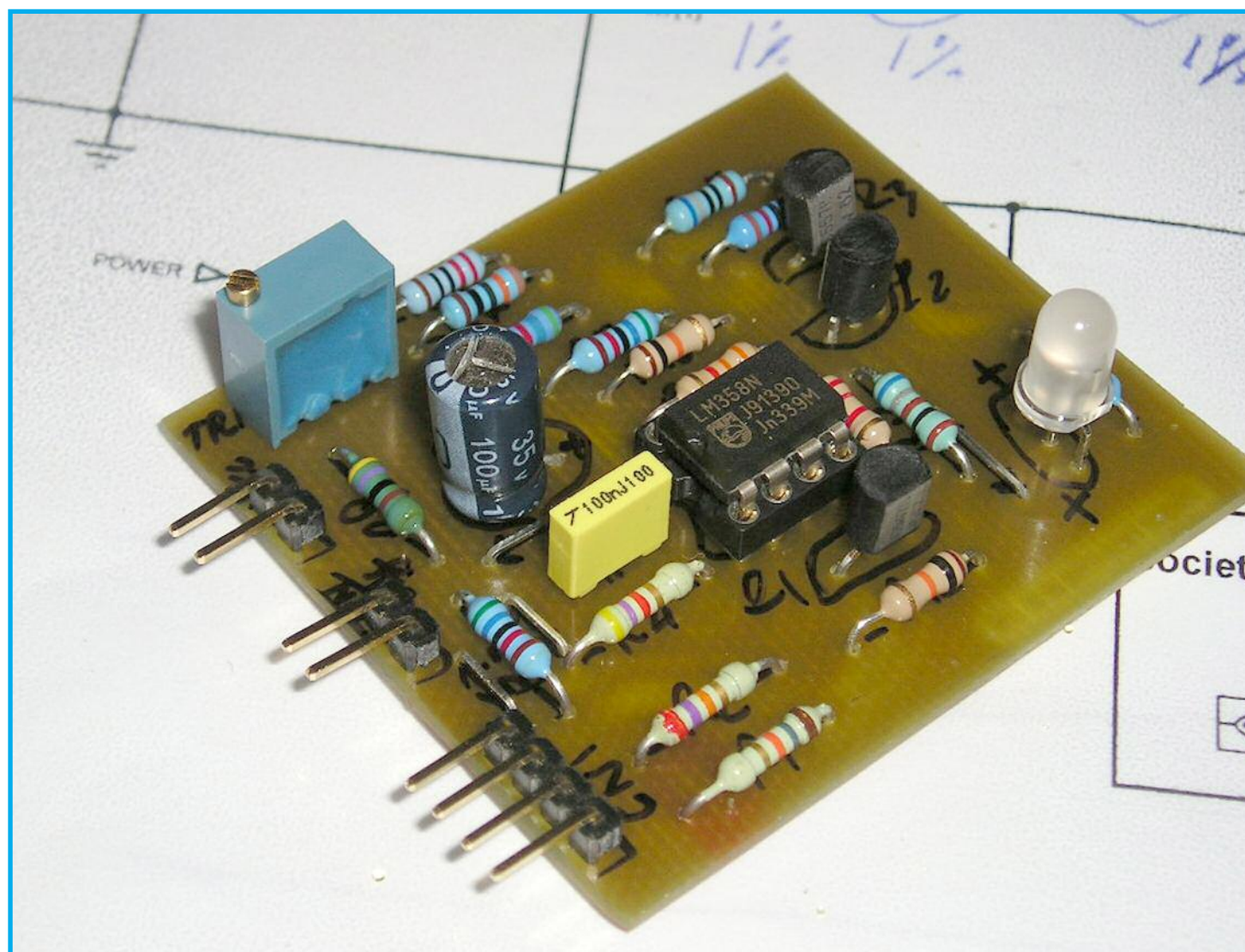
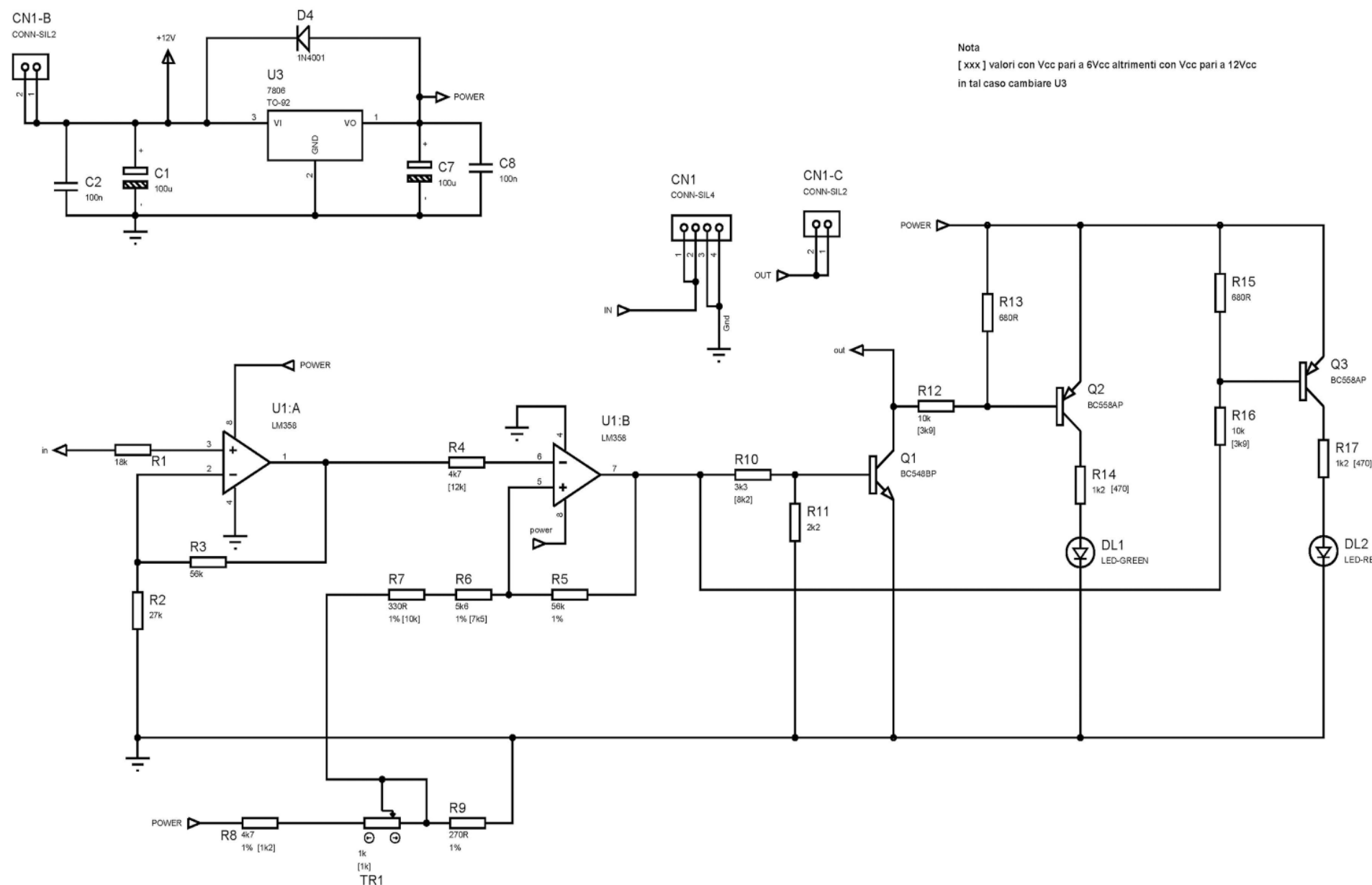



Figura 1: Schema elettrico della sezione di protezione dell'alimentatore. I due amplificatori operazionali sono utilizzati per amplificare la tensione in ingresso e come schmitt trigger.

cheremo con V_b) in modo che venga garantito ''lo scatto" alle tensioni di soglia prestabilite. La R_4 ha lo scopo di bilanciare la corrente fra i rami d'ingresso dell'Op.Amp. Lo stesso scopo è svolto dalla R_1.

Il doppio operazionale utilizzato è un LM358 in package DIP-8 mentre i tre BJT sono un BC548 e due BC558 in TO-92.

Per un'erogazione di corrente inferiore alla massima, la protezione non interviene.

In queste condizioni l'Op.Amp U_1:B ha il piedino d'uscita (pin 7) alla tensione V_{sat} ovvero circa a 5,5V quindi, Q_1 è in profonda saturazione. Il secondo BJT Q_2 sarà saturo e permetterà l'accensione del led DL_1. Q1 avrà il collettore ad un potenziale molto prossimo allo zero. Essendo connesso direttamente sul piedino 5 dell'LM2576,lo porterà ad un potenziale basso permettendo all'alimentatore di poter funzionare regolarmente.

Il transistore Q_3 sarà interdetto quindi, il led DL_2 risulterà spento. Si fa notare che i led DL_1 e DL_2 nella pratica corrispondono ad un doppio led con catodo comune.

Si guardi la **foto 1** per una miglior comprensione dei componenti. Sulla destra è visibile il led, a sinistra abbiamo il trimmer TR_1 mentre Q_2 e Q_3 sono in alto.

Nella **foto 2** è possibile notare le ridotte dimensioni della protezione elettronica paragonate con due nuclei impiegati nel progetto. Il PCB è stato particolarmente curato in modo da evitare disturbi dati dal campo magnetico nei pressi dell'induttore. Allo scopo si è deciso per realizzare un piano massa generale a protezione delle piste di segnale.

Il calcolo dei componenti

La R_{19} impiegata nello schema di figura 1 (si veda la prima parte) svolge la funzione di sensore di corrente ed ha un valore di 0,47Ω.

La **foto 3** mostra il componente a ridosso del modulo di protezione fissato il PCB principale dell'alimentatore.

Quando ai capi della R_{19} sono presenti 0,564V, la corrente sul carico è di 1,2A. Questa tensione è la tensione di soglia che, amplificata tre volte (G=3) dal primo Op.Amp. U_1:A, invia una tensione di 1,692V al piedino invertente del secondo Op.Amp. (pin 6).

La soglia V_H dello Schmitt Trigget è stata impostata a 1,6V quindi, la tensione d'uscita del secondo Op.Amp. si porta re-

foto 2: lato rame dello stadio di protezione elettronica comandato dall'IC LM358 . Si noti la presenza del piano massa. Più in basso, è possibile vedere il toroide della serie High Flux della Magnetics® e della Arnold

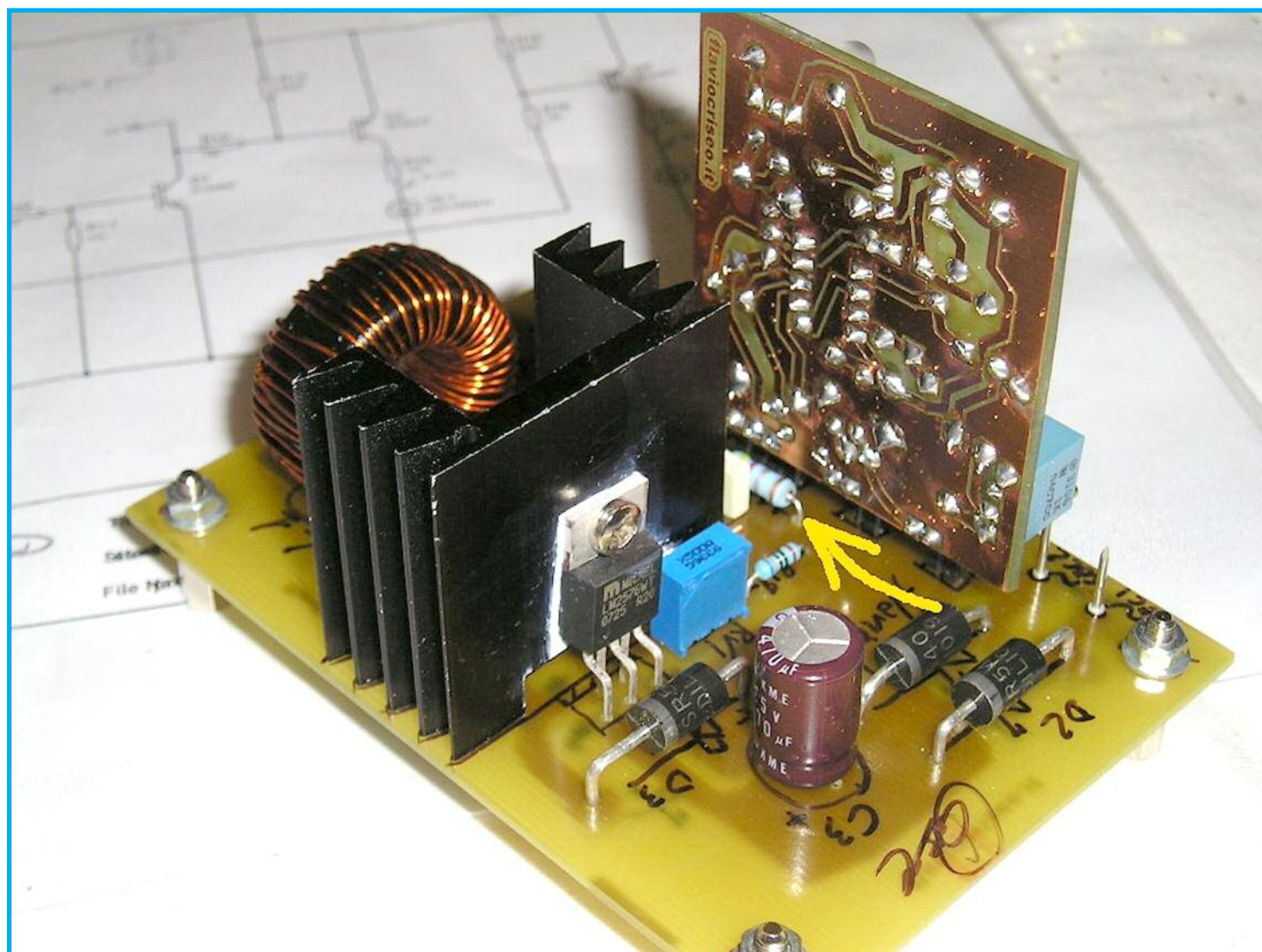



Foto 3: Nella foto è presente lo stadio di protezione montato sull'alimentatore. Si noti la freccia in giallo che indica la R_{-sense} avente il compito di sensore di corrente siglata R_{19} nello schema di figura 1

pentinamente dal valore ''sat'' al valore d'interdizione prossimo a zero Volt.
Questo porta all'interdizione di Q_1 e di Q_2 ed alla saturazione di Q_3 che permette l'accensione del led DL_2 di colore rosso.
Il led rosso quindi, indica un sovraccarico/cortocircuito mentre il led verde indica regolare funzionamento.
Compreso il funzionamento di massima sullo stadio di protezione, vediamo come si debba procedere per il calcolo dei componenti.
Abbiamo stabilito che il trigger di schmitt debba avere una V_H di 1,6V a fronte di una corrente di 1,2A.

Decidiamo la soglia minima d'intervento del trigger per una V_L di 0,6V quindi il ΔV d'isteresi è pari a 1V.
Si consideri adesso la **figura 2** ove è rappresentato in modo semplificato o stadio trigger ed i componenti principali. La tensione V_b nonché la tensione V_{ref} sul nodo **K** sono legate dalla seguente relazione ottenuta dalla sovrapposizione degli effetti.

$$V_{ref} = V_{out}\frac{R_1}{R_1+R_2} + V_b\frac{R_2}{R_1+R_2} \qquad (21)$$

La **figura 3** mostra la finestra d'isteresi pari ad 1Volt ottenuta con il trigger di schmitt progettato. E' possibile vedere come, per

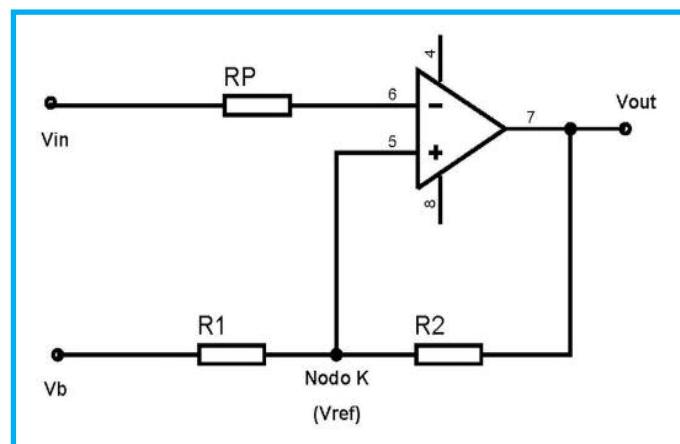



Figura 2: Schema semplificativo dello schmitt trigger. Si noti come la V_{in} posta sul piedino invertente determina lo ''scatto'' dell'Op.Amp. I valori di soglia sono visibili nella figura 3

valori della V_{in} inferiori alla tensione V_H, l'uscita V_{out} dell' Op. Amp. tenderà alla sat+. Tale valore non cambierà fino a quando la tensione V_{in} in ingresso al piedino invertente scenderà sotto i 0,6V. Tale valore corrisponderà ad una corrente sul carico pari a 700mA.
Nelle due condizioni di funzionamento, la tensione V_{ref} potrà avere due valori ben precisi ossia V_H e V_L. Applicando la (21) la V_{out} assumerà i valori Vsat+ e Vsat- e la V_{ref} (sul nodo K) si porterà alle suddette tensioni:

$$\begin{cases} V_H = V_{sat}\frac{R_1}{R_1+R_2} + V_b\frac{R_2}{R_1+R_2} \\ V_L = V_b\frac{R_2}{R_1+R_2} \end{cases} \qquad (22)$$

Un vincolo importante da rispettare nel dimensionamento delle resistenze sul ramo non invertente del trigger di schmitt è il seguente

$$R_2 >> R_1 \qquad (23)$$

risolvendo il sistema (22) e calcolando ri-

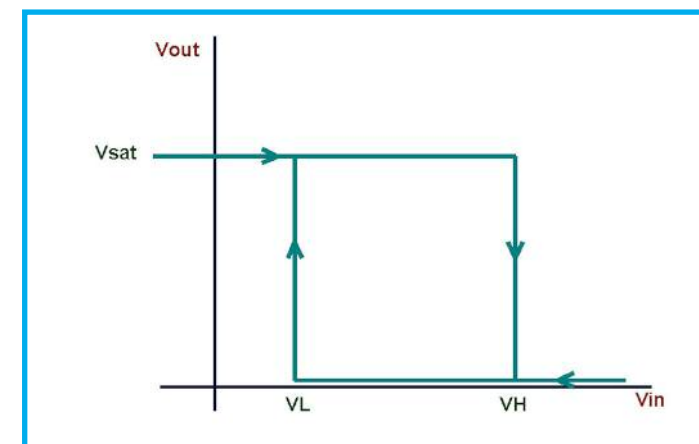



Figura 3: Finestra d'isteresi data dallo schmitt trigger utilizzato nello stadio di protezione elettronica.

spetto ad R_1 si ha che

$$R_1 \frac{R_2\Delta V}{V_{sat}-\Delta V} \qquad (24)$$

dove il termine ΔV è la finestra d'isteresi pari a 1V.
Per R_2=56kΩ e una tensione sat di 5,5V si ha che R_1 dovrà valere 17500Ω ovvero sarà data dalla serie fra un resistore da 10kΩ e uno da 7k5Ω (nello schema di figura 1 sono R_7 ed R_6 rispettivamente e corrispondono a R_1 di figura 2).
A questo punto si rende necessaria la determinazione della V_b che consentirà d'avere le soglie V_H e V_L desiderate.
Sempre dalla seconda equazione del sistema (22) si ricava il valore necessario per la V_b che risulta essere circa 0,663V.
Il partitore di tensione da dimensionare visibile in figura 1 è dato da R_8 TR_1 R_9.
Fissando una corrente di 2mA possiamo dimensionare la R_9 nel modo seguente:

$$R_9 = V_b/I = 330 \text{ Ohm} \qquad (25)$$

mentre la somma fra R_8 e TR_1, che chia-

meremo R_x, deve darci

$$R_x=(V_{cc}-V_b)/I=2168 \text{ Ohm} \qquad (26)$$

si può quindi decidere per i seguenti valori

$$R_8 = 1k2\Omega,\ TR_1=1k\Omega,\ R_9=220\Omega.$$

In questo modo avremo la possibilità di tarare con maggior precisione il partitore.
I componenti fin qui dimensionati ed impiegati per lo schmitt trigger sono a film metallico con ±1% di tolleranza e visibili in foto 1.

TEST, MISURAZIONI, COLLAUDI E NOTE TECNICHE SUL LAYOUT

Come in tutti i progetti che si rispettino, la fase di collaudo e test è uno step d'obbligo.
Innanzitutto e importante verificare che la corrente massima erogata sia quella da noi stabilita.
Successivamente è necessario verificare che tale corrente sia erogata anche in presenza del circuito di protezione al sovraccarico da noi previsto.
Fatto ciò, si passa alla verifica dell'intervento della protezione elettronica verificando il valore d'intervento nonché l'affidabilità d'intervento.
Durante i test, dev'essere sempre tenuta sotto controllo la temperatura nonché le forme d'onda più importanti in modo da verificare se l'alimentatore si trovi in condizioni accettabili di funzionamento.
Per effettuare i primi collaudi e test ci si connette all'alimentatore così come nella **foto 4.**

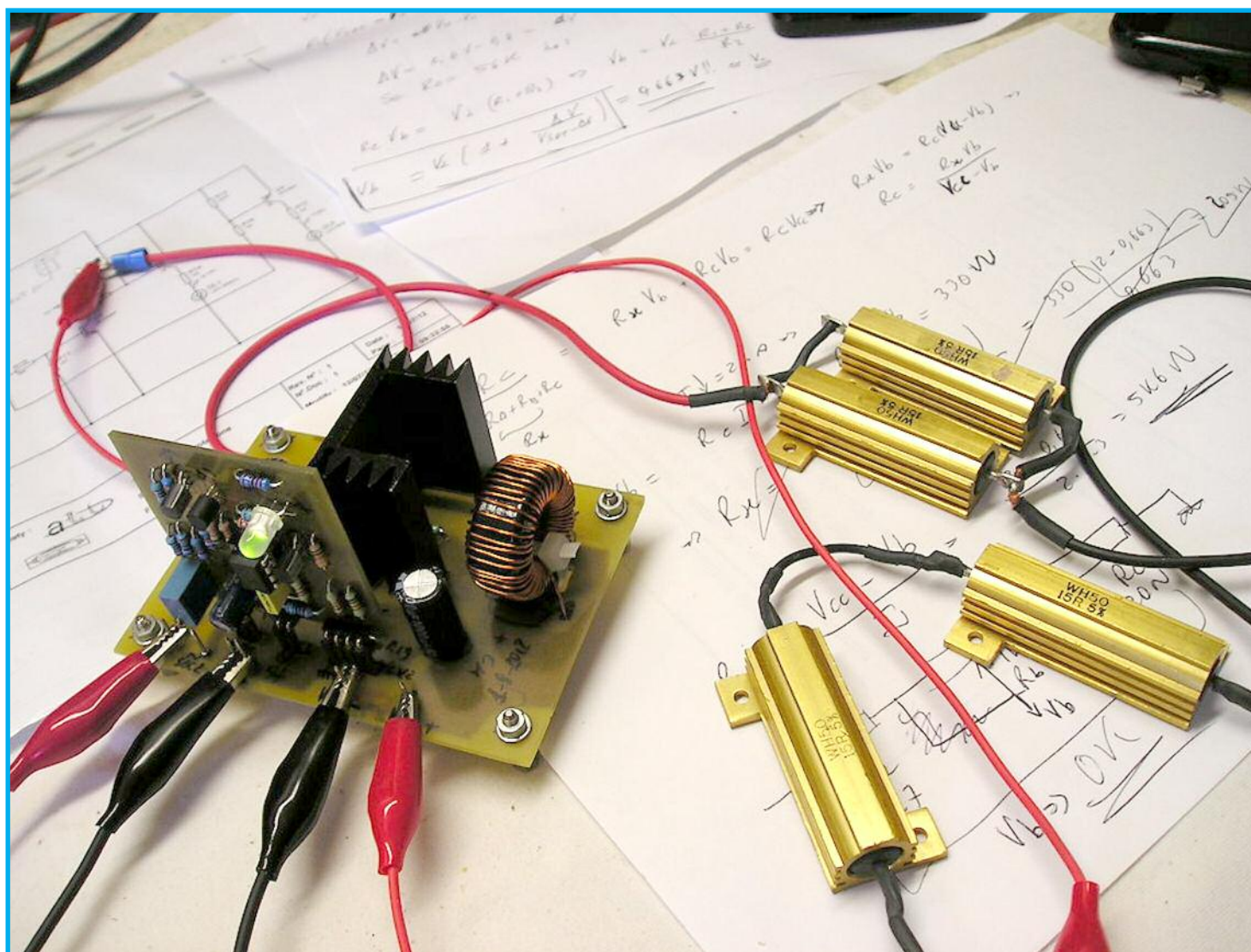



Foto 4: Durante il normale funzionamento, il doppio led DL_{1-2} è di colore verde. Il carico da noi utilizzato per i test preliminari è consistito in alcuni resistori corazzati da 15Ω

A seconda della corrente che si vuol far erogare si connettono, in uscita all'alimentatore, due resistori corazzati da 15Ω fra loro in serie o in parallelo e si alimenta il tutto con batteria al piombo gel da 12V.
Con la coppia di resistori in parallelo si ha una corrente di circa 960mA mentre con al coppia di resistori in serie di ha una corrente di 240mA.
Ci si collega con oscilloscopio sul pin 2 dell'LM2576 e subito a valle dell'induttore L_1.
Nella **figura 4** il carico è da 30Ω. La traccia numero 1 è relativa alla sonda oscilloscopica connessa in DC-mode con 5V su divisione mentre in basso a destra è possibile vedere come la frequenza di commutazione sia di circa 53kHz ovvero prossima ai 52kHz tipici dell'IC.
La fase di ''on" del BJT interno all'integrato LM2576 determina nell'uscita (piedino 2) un'onda quadra di circa 14V. Poiché il carico è di natura statica, abbiamo un duty cicle fisso. Per tutto il tempo in cui si ha la fase di ''on" la rampa di tensione tende la suo valore di cresta.
Il secondo canale dell'oscilloscopio è connesso in AC-mode e la sonda è posta in uscita direttamente a valle dell'induttore principale.
La forma d'onda del secondo canale permette d'osservare il ripple (talvolta conosciuto come ''distorsione sovrapposta") che si ha nell'intorno della tensione d'alimentazione continua stabilita sui $7{,}2V_{dc}$.
Il suo andamento mostra l'energia accumulata nell'induttore mentre, nella fase discendente del ripple (fase di ''off" dell'integrato) l'energia immagazzinata dall'in-

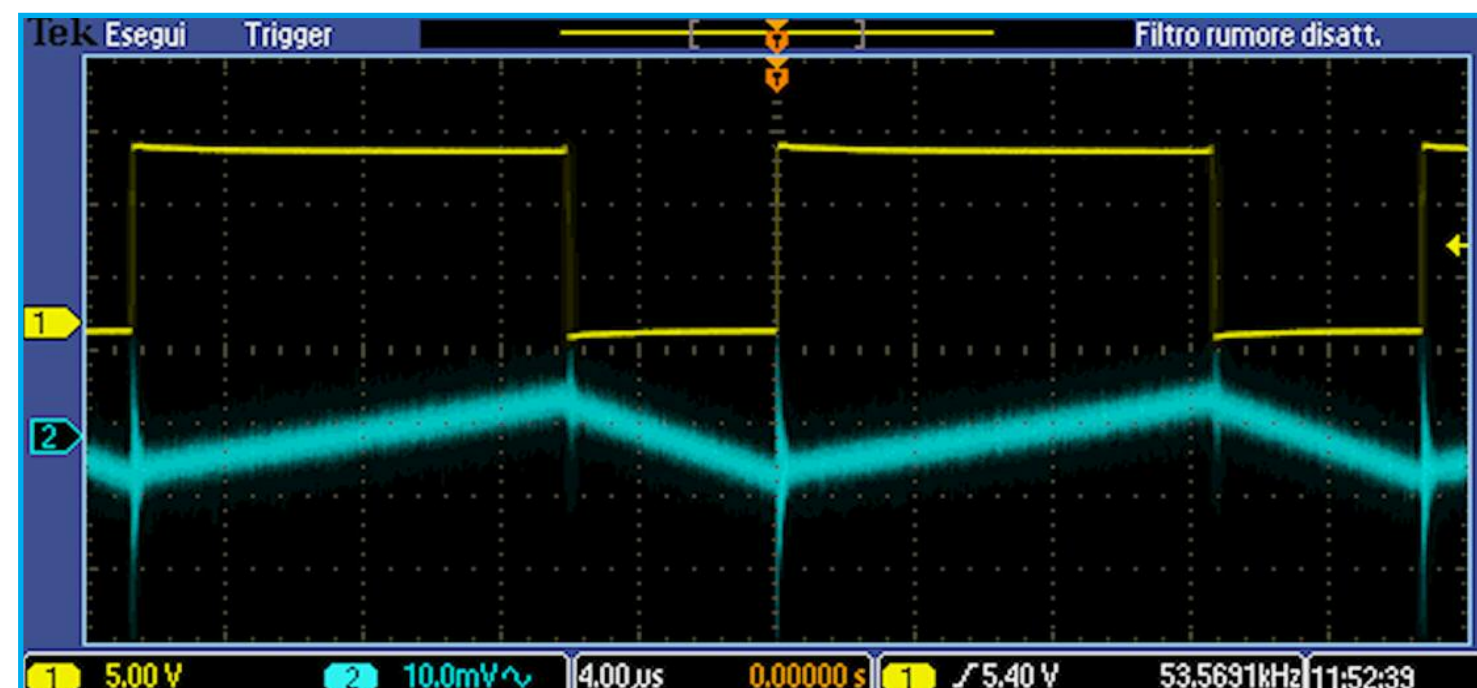



Figura 4: Gli oscillogrammi presenti nella figura sono stati prelevati dall'uscita dell'IC LM2576 e a valle dell'induttore L_1. Si noti come la frequenza di switching è leggermente più alta rispetto al valore nominale da 52kHz. Ciò è dato dal fatto che il carico è molto esiguo (240mA circa) rispetto al valore massimo previsto.

duttore diminuisce perché è ceduta al carico.
Mentre sul canale uno è lecito aspettarsi un'onda quadra con un determinato duty cicle, nel secondo canale (connesso in AC-mode) è lecito aspettarsi la classica onda a dente di sega così come mostrato in figura 4.
Il rapporto tra il valore efficace dell'onda a dente di sega e il livello di tensione continua rilevata sul carico ci danno l'informazione relativa alla distorsione.
Minore è tale rapporto, maggiore sarà la qualità della tensione d'uscita e quindi dello stesso alimentatore.
Questo valore deve essere verificato sia in presenza di minimo carico che nelle condizioni di massimo carico (condizione più sfavorevole).
Nel nostro caso, in base alle misure sperimentali effettuate, la distorsione sovrapposta è pari al

$$D = \frac{V_{pp}}{2\cdot\sqrt{3}\cdot V_{out}} = \frac{15\cdot 10^{-3}}{2\cdot\sqrt{3}\cdot 7{,}2}\cdot 100 \cong 0{,}06\% \quad (27)$$

La notevole qualità della tensione V_{out} è determinata, oltre che da un layout molto accurato, dalla qualità della capacità C_4 in uscita. Qualora si volesse diminuzione ulteriormente il ripple, si potrebbe inserire un ulteriore gruppo LC così come detto in precedenza (vedi prima parte).
Rispetto al prototipo, il modello commerciale da noi analizzato in precedenza e visto in foto 1, presenta invece un ripple nell'intorno del 15%. I motivi possono essere molteplici: dal layout con piste lunghe e sottili ad una eccessiva ESR troppo alta nella capacità d'uscita.
Il costo del condensatore impiegato è certamente inferiore a quello da noi adottato.

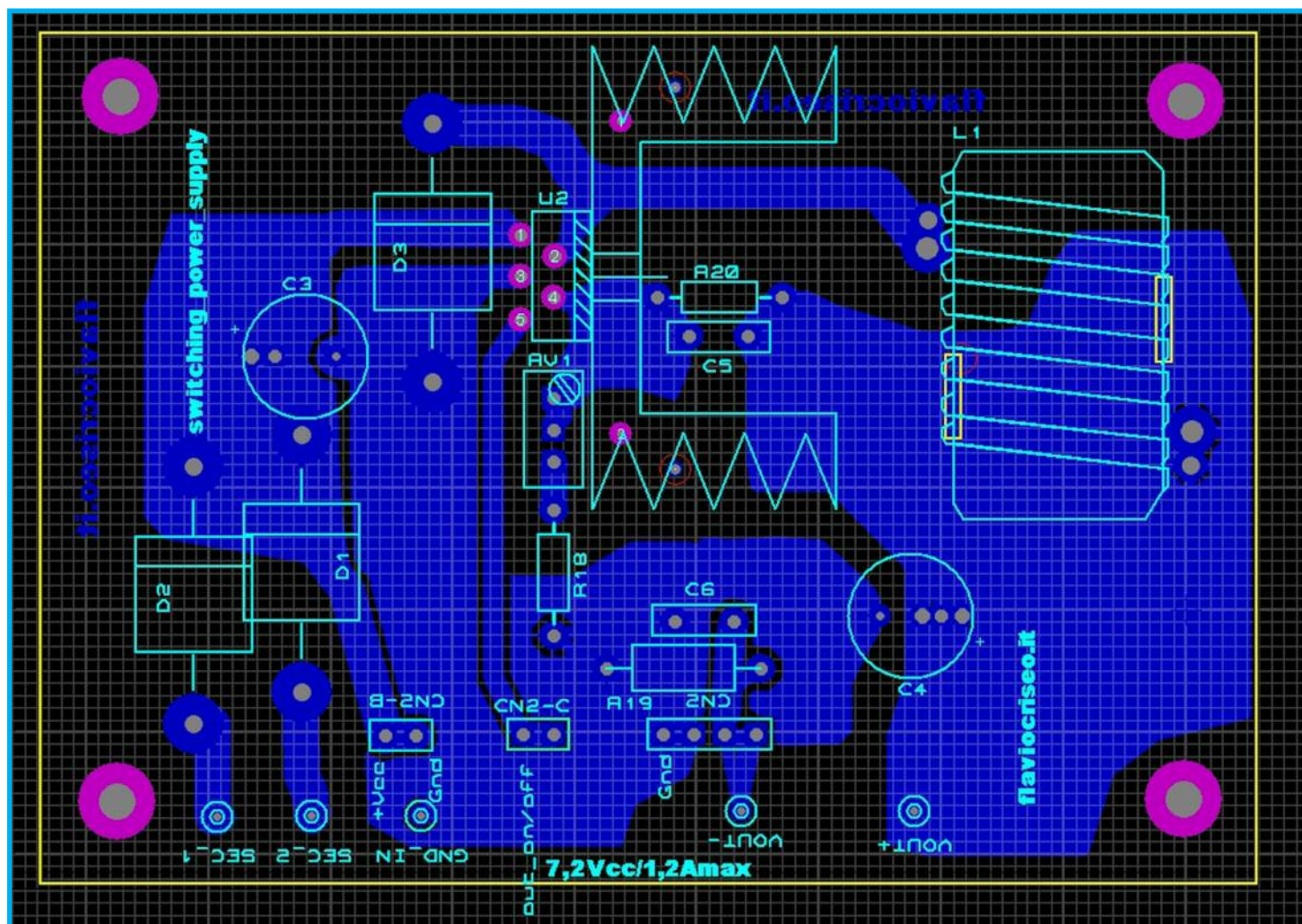



Figura 5 : Layout dell'alimentatore switching da 1,2A. Si notino come le piste di rame verso massa sono molto larghe ed inoltre, le connessioni dei componenti principali (diodo D_3, Capacità C_3 e C_4) sono quanto più vicine possibili all'IC principale

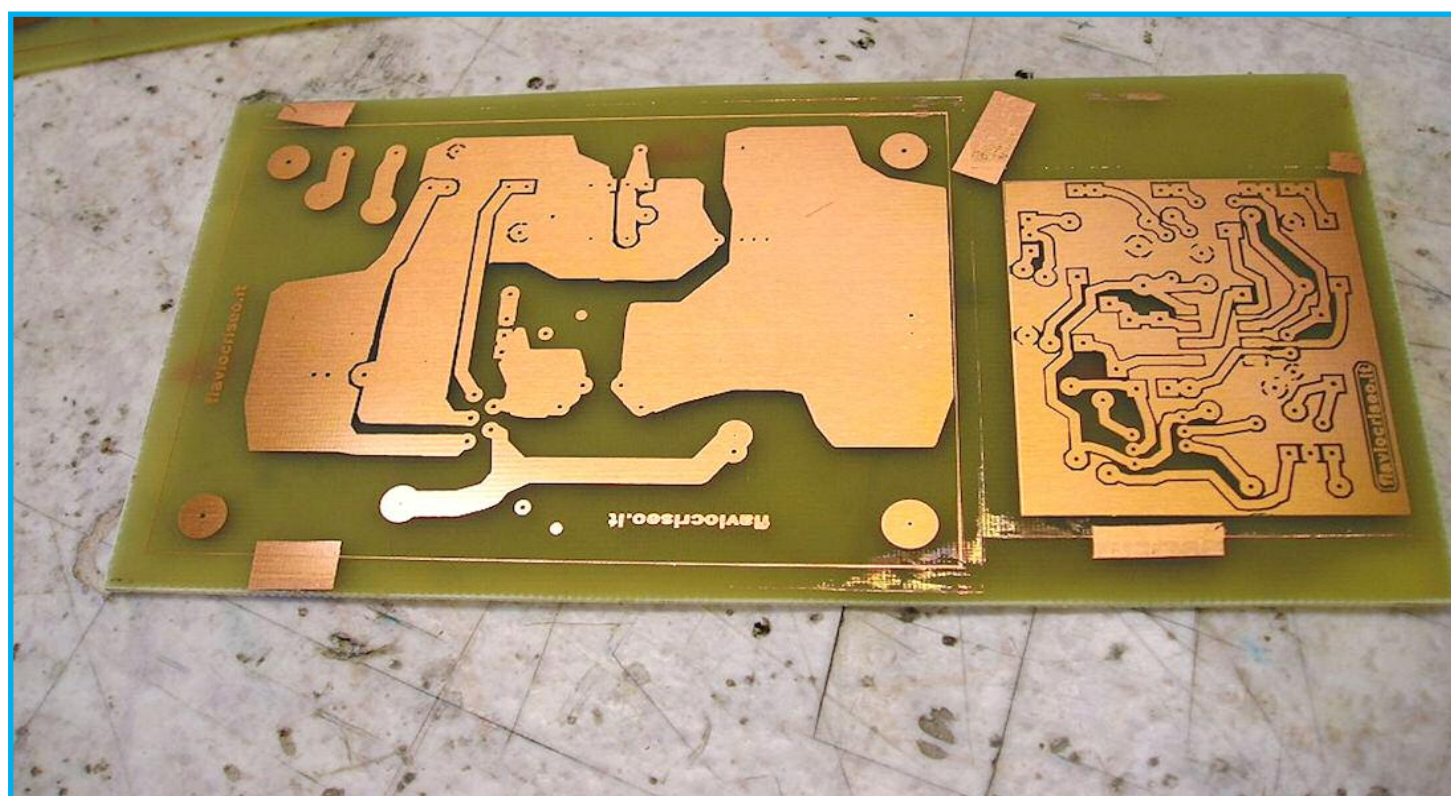



Foto 5 : La foto evidenzia i circuiti stampati dopo l'incisione con cloruro ferrico. È importante accertarsi che le piste siano ben definite, non presentino angoli troppo "spigolosi" e quanto più larghe possibile

Per gli alimentatori switching, lo studio del layout risulta essere determinante in quanto il peggiore pericolo sono le induttanze parassite e le capacità d'accoppiamento indesiderate. Per risolvere questi problemi, è necessario realizzare piste larghe e corte ed inoltre connettere quanto più vicino possibile fra loro alcuni componenti importanti. A tale riguardo è possibile vedere la **figura 5** e la **foto 5**.
Sempre nella figura 4, è possibile notare come il dente di sega presenta degli spike negli istanti in cui si ha il passaggio fra on-off e viceversa. Ciò è normale ed è causato da fenomeni parassiti che nel brevissimo periodo evidenziano il gradiente di tensione.
È importante avere una buona idea dell'entità di questi spike perché questi possono essere causa di guasti o cattivi funzionamenti sia per gli interruttori dipendente ed indipendente che per i componenti passivi principali.
Lo zoom di **figura 6** mette in evidenza l'entità di questi spike ed in particolar modo sul fronte di salita dell'onda quadra (si veda anche la figura 4). Il fronte di salita mostra la fase di passaggio fra il T-off ed il T-on del BJT interno all'IC LM2576. Come si può notare nel tracciato 1 di figura 6, la rampa si salita è molto buona ed in oltre presenta solo una leggera oscillazione in corrispondenza del picco più alto dello spike. L'entità di quest'ultimo è nell'intorno dei $40mV_{picco\text{-}picco}$ quindi, del tutto accettabile.
Lo stesso test deve essere effettuato sull'alimentatore di **figura 5** ma, stavolta, con un carico prossimo al massimo consentito.

Ponendo i due resistori corazzati da 15Ω in parallelo, ne otteniamo uno da 7,5Ω capace di far erogare 970mA circa (ci poniamo quindi all'80% delle condizioni previste per il nostro sistema).

L'oscillogramma presente nella **figura 7** mostra come lo spike è sempre contenuto entro i valori precedenti ed inoltre il ripple è, com'era d'aspettarsi, leggermente maggiore del precedente ovvero nell'intorno dei 20mV (D=0,08%).

Possiamo vedere come la **figura 8** permette uno zoom accurato dello spike.

La sovraoscillazione relativa alla traccia 2 ne mostra come si possa arrivare a circa 75mV$_{picco-picco}$ quindi un incremento dello spike pari a circa l'87% della precedente condizione di funzionamento. La traccia numero 1 moltra come la rampa di salita sia sempre buona ma con una oscillazione più marcata. È possibile stimare il tempo d'assestamento nell'intorno degli 80ns a fronte dei 50ns della risposta precedente.

Lo spike visualizzato in questo caso è comunque sempre accettabile.

Spostando la sonda numero due a monte dell'integrato LM2576 (piedino 1) è possibile vedere quali possano essere gli stress subiti dalla batteria d'alimentazione nonché da un eventuale trasformatore posto in ingresso. Questa misura ci da inoltre un'idea relativa allo stress alla quale sono sottoposti i diodi D_1 e D_2 .

Una panoramica relativa a quanto appena detto, è visibile in **figura 9** ove lo stress al quale sono sottoposti i diodi e la sorgente di tensione V_{in} è visibile in modo agevole (si veda la traccia 2 dell'oscillogramma).

Volgendo adesso lo sguardo alla **figura 10** notiamo uno zoom relativo alla figura 9. si vedono dei picchi in ingresso durante il passaggio fra Ton e T-off ad opera dello switching.

Il picco in ingresso è sempre modesto, infatti abbiamo un valore massimo di circa 350mV$_{picco-picco}$ con carico da 7,5Ω.

figura 6: Fronte di salita relativo al passaggio fra la fase di T-off e la T-on. La traccia 2 indica l'andamento della tensione in uscita a valle dell'induttore L_1

Figura 7: Ponendo un carico da 7,5Ω in uscita, l'andamento della tensione su pin dell'LM2576 e sulla V_{out} sono le seguenti. Si noti come il ripple è davvero contenuto

Figura 8: A fronte di una richiesta di corrente dell'ordine delle 970mA, la rampa di salita in fase di passaggio all'istante T-on è molto buona. Nella traccia 2 si noti come le oscillazioni sono leggermente più marcate rispetto alla figura 6. Ciò è dato dalla differente corrente richiesta

Passando alle misure relative al prototipo con l'induttore di maggiori prestazioni (il modello 77930-A7 Kool Mμ) possiamo vedere come le **figure 11** e **12** presentano un ripple molto contenuto in uscita ed ''uno stress" di commutazione in ingresso sempre nell'ordine delle $400mV_{picco-picco}$.
La figura 11 è relativa a un carico da 30Ω mentre la figura 12 evidenzia lo stress che si ha in ingresso quando il prototipo è connesso ad un carico da 7,5Ω (si veda la foto 4).
Il ripple del prototipo impiegante l'induttore 77930-A7 è davvero contenuto, la **figura 13** infatti, evidenzia un'onda a dente di sega inferiore ai $10mV_{picco-picco}$ quando è connesso al carico da 7,5Ω.

Figura 9: Le forme d'onda in ingresso mostrano come la non linearità dello switching crei distorsione anche in ingresso.

Figura 11: Impiegando l'induttore con nucleo 77930-A7 il ripple in uscita è davvero contenuto. La qualità della tensione d'uscita è davvero eccellente.

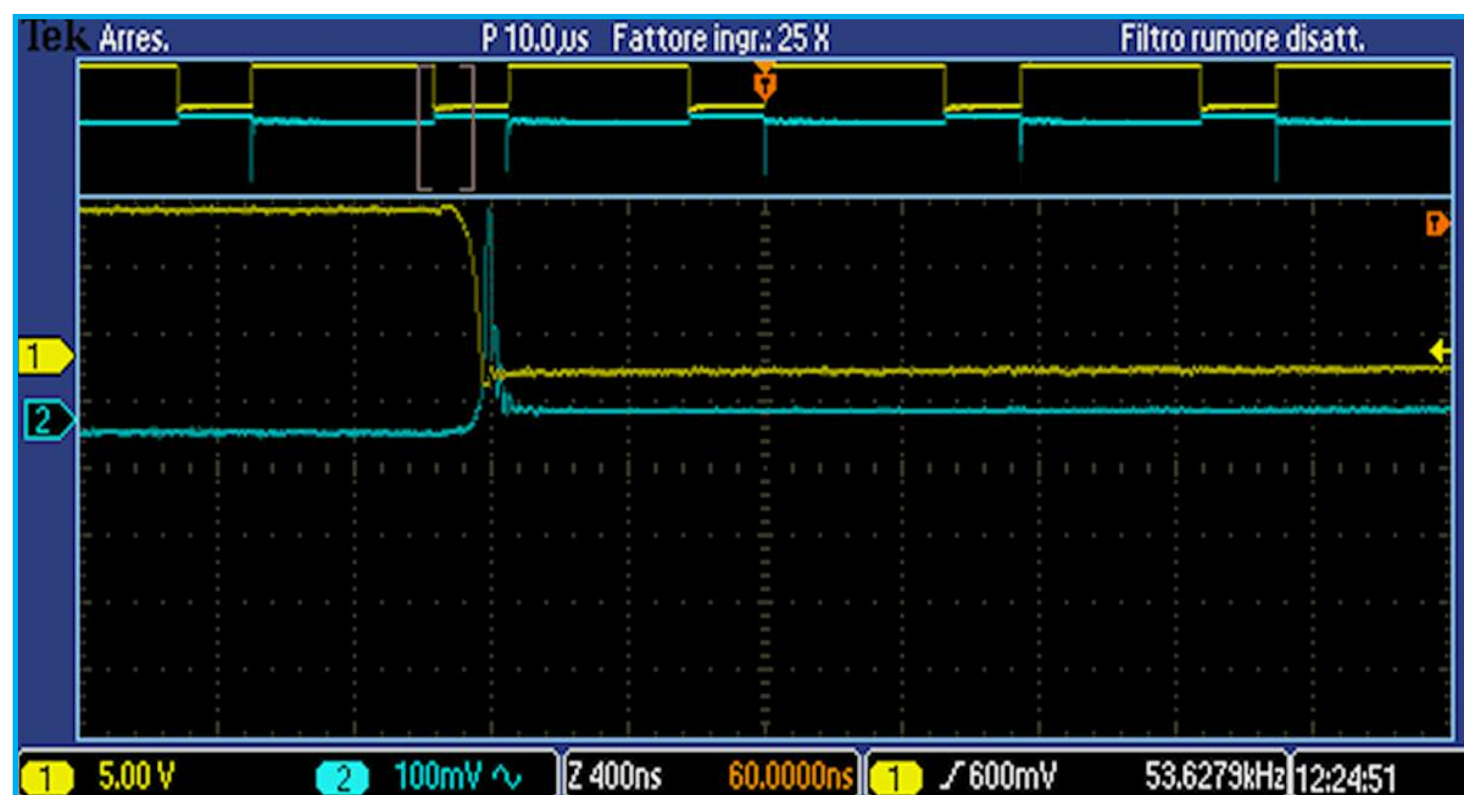



Figura 10: Zoom relativo al picco di distorsione in ingresso durante il gradiente di tensione fra il T-on ed il T-off

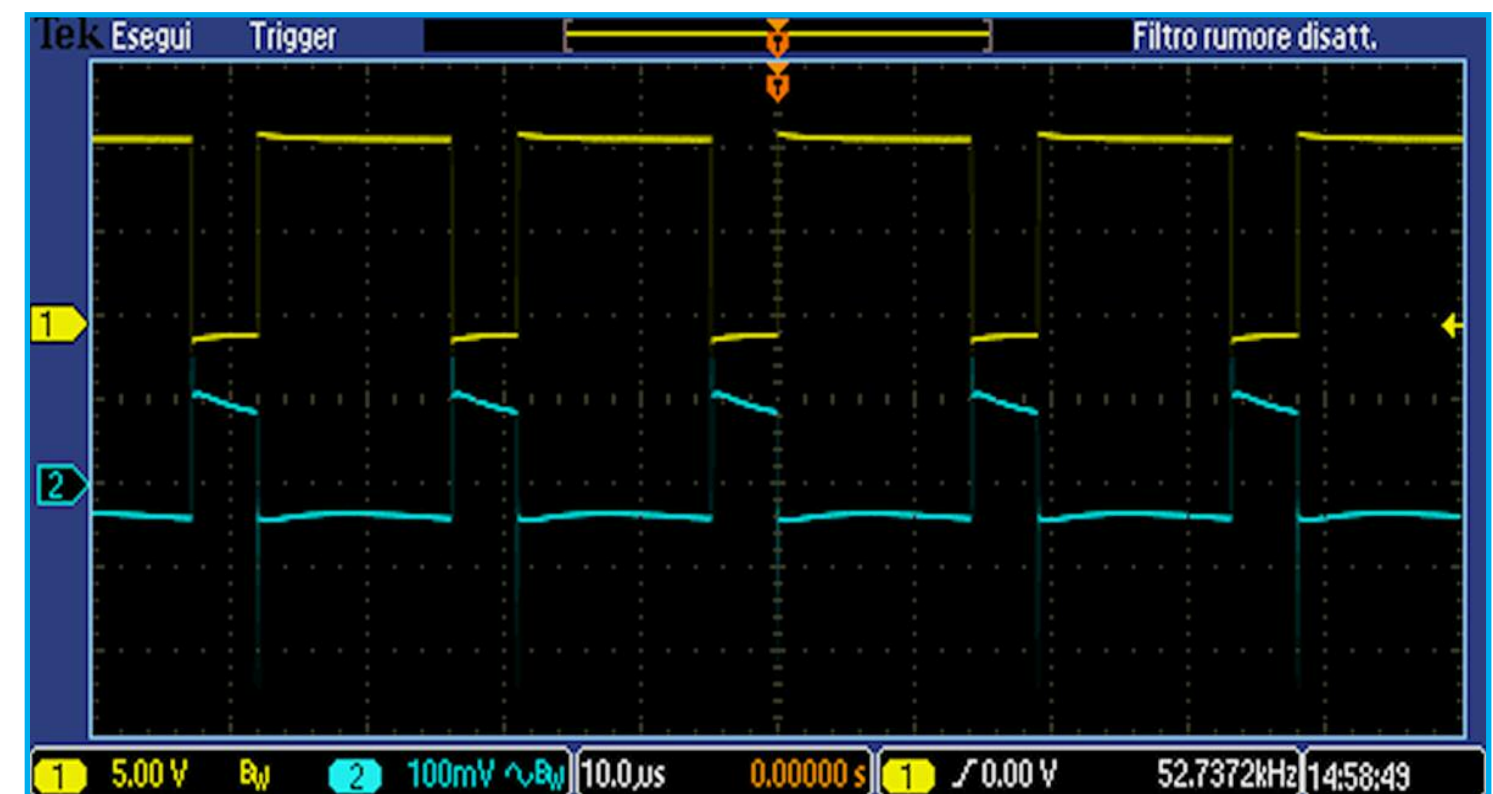



Figura 12: A fronte di una corrente di circa 970mA erogata sul carico, si ha una più marcata distorsione della tensione in ingresso al pin 1 dell'IC LM2576

Forme d'onda in corrente e funzionamento a vuoto ed in corto

Tornando nuovamente sulla figura 5 è possibile vedere come il PCB permetta agevolmente l'interruzione della pista che connette l'IC all'induttore L_1.
In questo modo, è possibile inserire un resistore antiinduttivo da 1Ω in modo da porre ai suoi capi la sonda dell'oscilloscopio.
le **figure 14** e **15** mostrano l'andamento della corrente, con un carico da 30Ω, nel prototipo con l'induttore 77930-A7 (figura 14) e nel prototipo con induttore 58206-A2 (figura 15) dopo la connessione dell'oscilloscopio ai capi del resistore da 1Ω posto

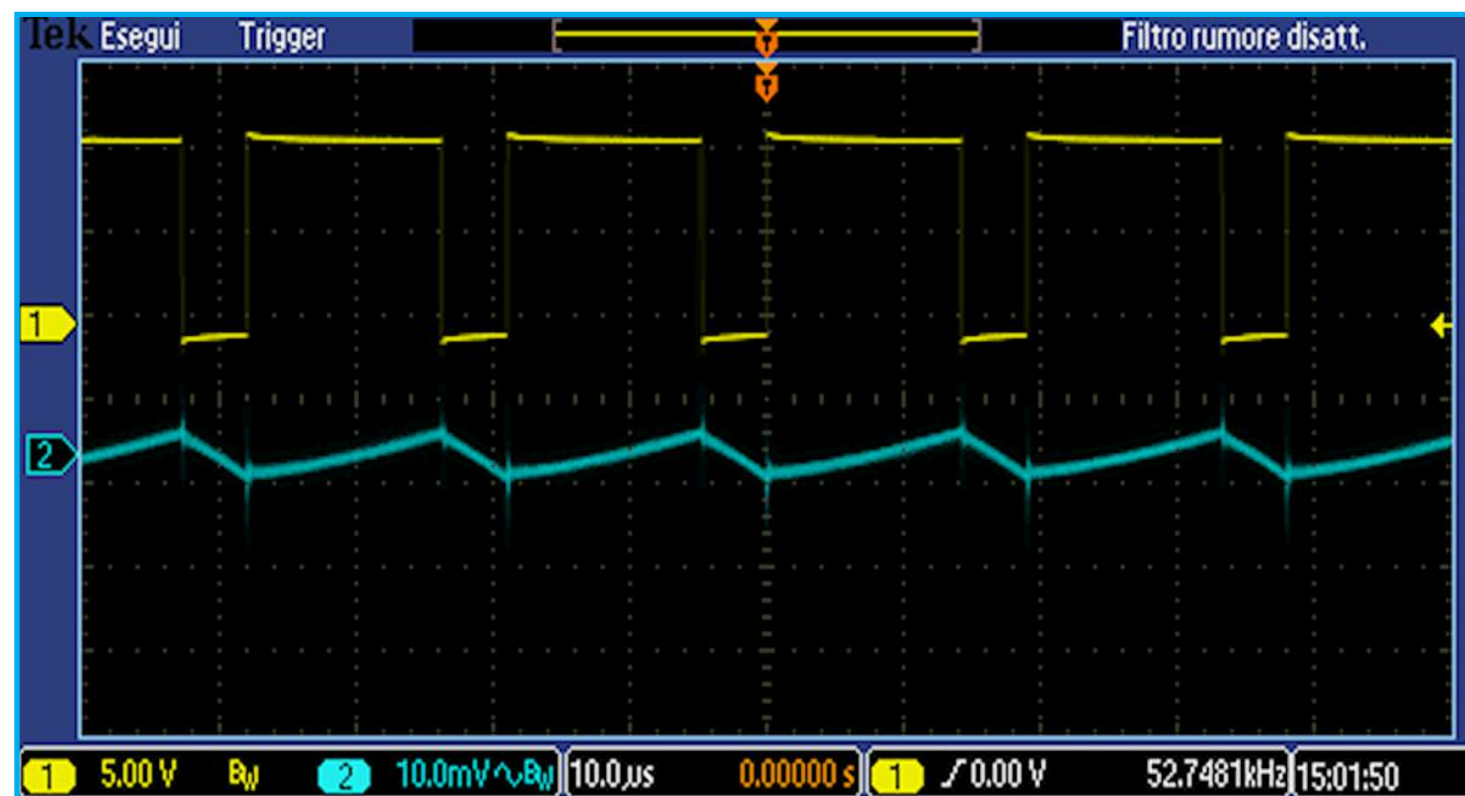



Figura 13: Con l'induttore Kool Mµ ed una capacità in uscita avente low_ESR da 1000µF, il prototipo presenta ottime prestazioni in termini di ripple

figura 14: Andamento della corrente sull'induttore con carico a 30Ω e toroide 77930-A7. È necessario interrompere la pista fra il pin 2 dell'IC e l'induttore L_1, connettere un resistore antiinduttivo da 1Ω e porre la sonda oscilloscopica ai suoi capi.

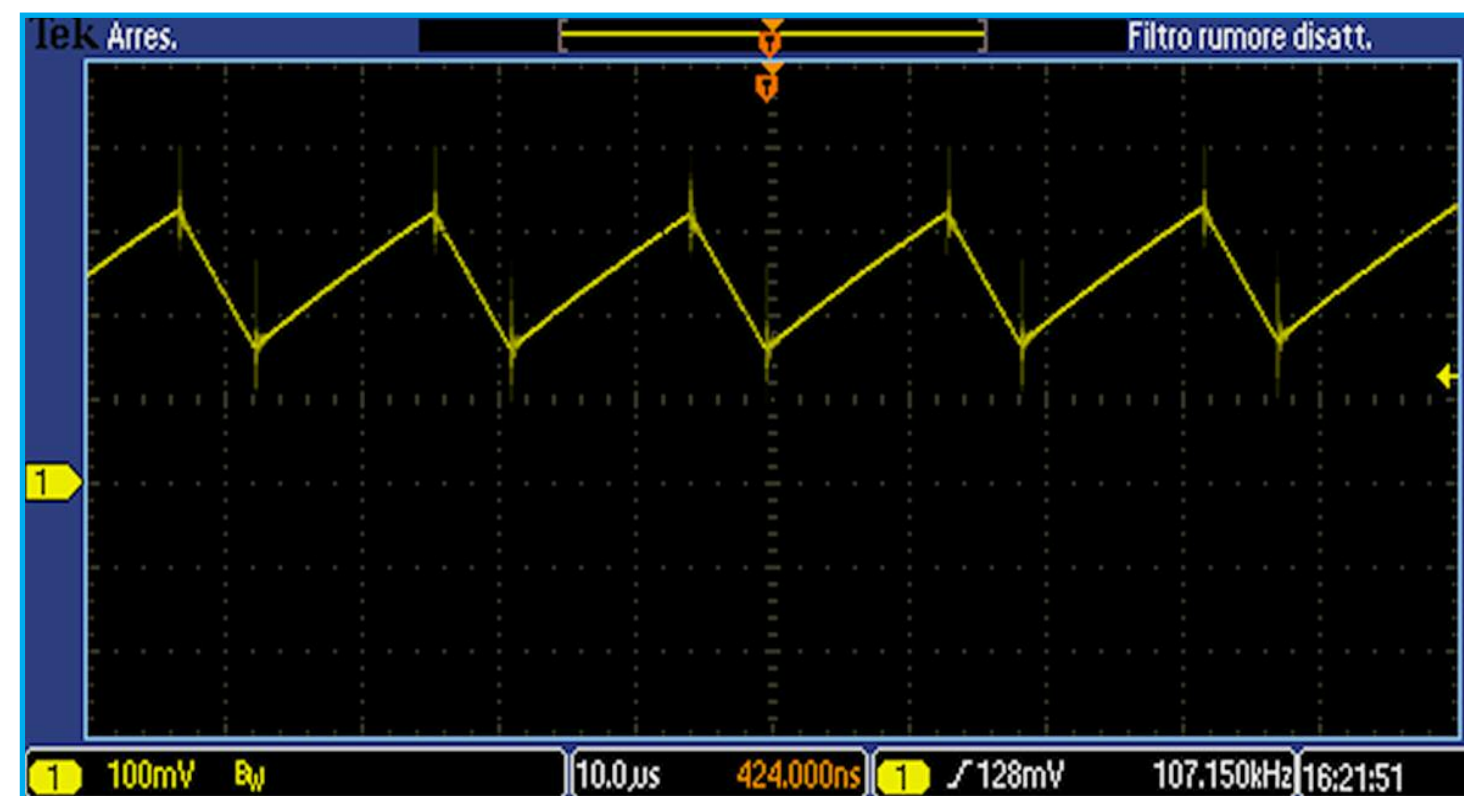



figura 15: Andamento della corrente sull'induttore con carico a 30Ω e toroide 58206-A2

figura 16: Andamento oscillante della V_{out} dato dall'intervento della protezione elettronica visibile in figura 1 (prima parte)

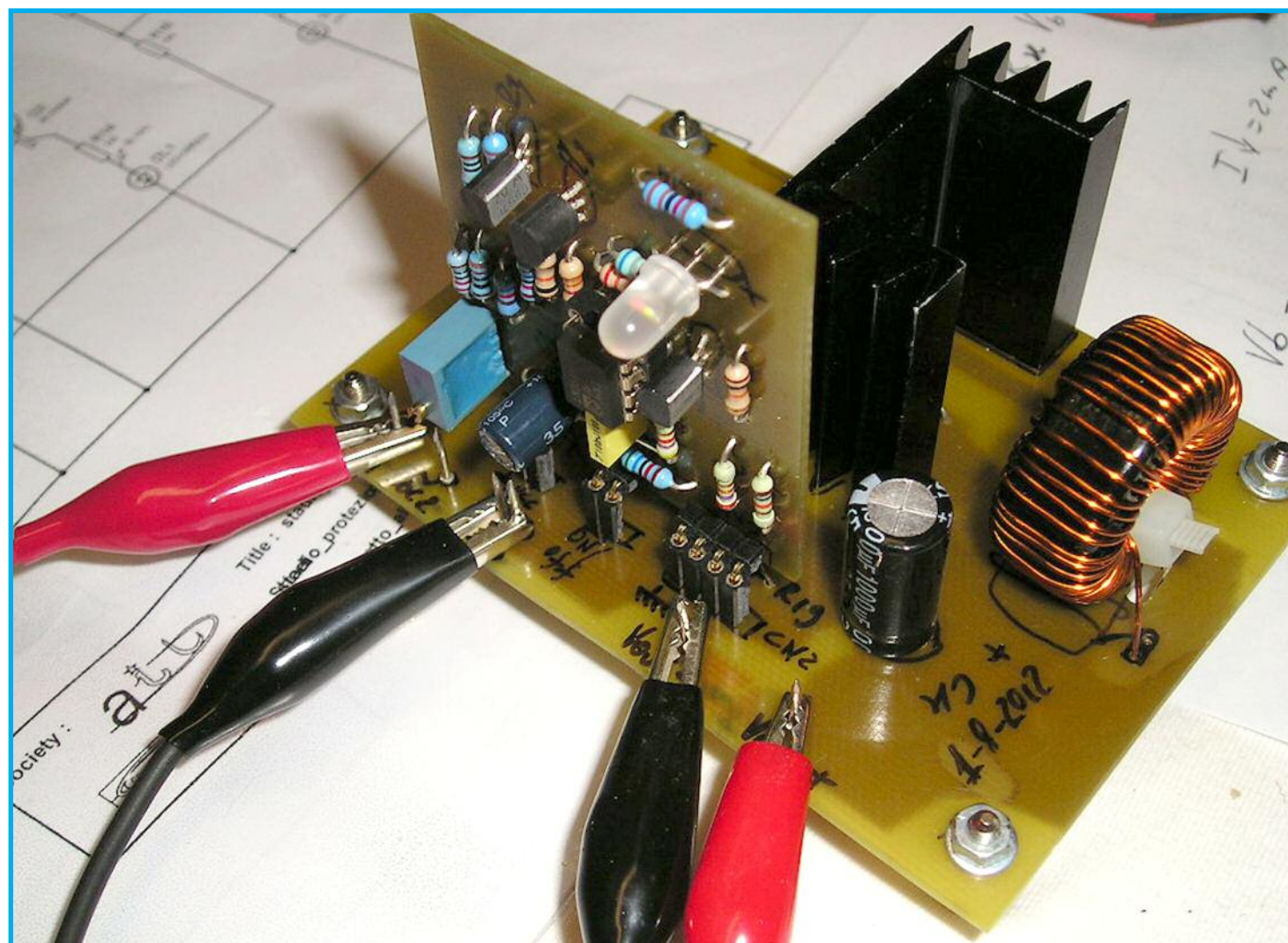



foto 6: In foto è possibile notare come in presenza di sovraccarico il doppio led DL_{1-2} accende alternatamente il filamento rosso e quello verde. Ciò è dato dallo "scatto" dello schmitt trigger ad opera di U_1:B

figura 17: Una condizione di funzionamento da evitare è data dall'assenza di carico quando lo stadio switching è funzionante. Le oscillazioni troppo marcate di figura evidenziano la condizione di funzionamento non ottimale per il BJT interno all'IC LM2576

fra il pin 2 dell'IC e l'induttore L_1.
A meno di una divisione dei tempi differente, gli andamenti sono molto simili e gli spike di corrente sono molto limitati.
Provochiamo adesso un forte sovraccarico in uscita e vediamo quale sa l'intervento della protezione elettronica.
La **figura 16** si può vedere come l'andamento della tensione V_{out} in presenza di cortocircuito sul carico, presenta un andamento oscillante tipico della carica e scarica di un gruppo RC.
Se facciamo riferimento alla figura 1 (vedi prima parte) si può notare come tale gruppo sia dato dalla R_{19} e da C_6.
La **foto 6** mostra il diodo led con entrambi i filamenti accesi relativi al rosso ed al verde.
Lo scatto fotografico è relativo alla presenza di un cortocircuito netto all'uscita dell'alimentatore.
Nella fase crescente della forma d'onda 2 di figura 16, si ha un picco di tensione (forma d'onda 1) all'uscita dell'IC. Questo è dato dal fatto che l'alimentatore tenta la partenza ma, la protezione elettronica comandata dal trigger di schmitt interviene.
Il pin 5 dell'IC è quindi posto a potenziale alto bloccando l'oscillazione dello stadio.
Quando i valori di corrente scendono sotto i 700mA lo schmitt trigger tenta nuovamente la partenza ma l'aumento della corrente sulla R_{19} fa sì che il ciclo di blocco si ripeta.
In definitiva, il doppio led tenta l'accensione verde per poi effettuare quella rossa.
Questa condizione di funzionamento permane fintantoché il sovraccarico/cortocircuito non è eliminato sulla V_{out}.
La dissipazione termica è equivalente alle condizioni di massimo carico quindi, dimensionando il dissipatore per tale condizione d'impiego non si avranno problemi.
Poiché il modello toroidale di maggiori dimensioni 77930-A7 è della serie Kool Mµ, ad orecchio non si sentono magnetostrizioni, mentre quando il sovraccarico è provocato sul prototipo con il modello High Flux 58206-A2 si hanno leggere magnetostrizioni.
Una condizione non molto raccomandabile di funzionamento è evidenziata in **figura 17** ove le forme d'onda sono relative alle tensioni sul pin 2 dell'IC e in uscita quando lo stadio non presenta alcun carico connesso.
Alimentando con 12V in ingresso si hanno oscillazioni smorzate molto forti. Per ovviare a tale problema sarebbe sufficiente connettere un piccolo carico permanente al sistema, ad esempio un led che ne indichi le condizioni di funzionamento.

Per approfondire

Per ulteriori note in merito all'utilizzo dei circuiti integrati trattati nell'articolo, è possibile cercare sul sito Texas Instruments la nota applicativa SNVC124B nonché il catalogo della Magnetics scaricabile su http://www.mag-inc.com/ per la scelta dei toroidi.

di VINCENZO SORCE

PROGRAMMARE IN ANDROID GOOGLE "APP INVENTOR"

Per chi intende sviluppare applicazioni per tablet o smartphome basati su Android, google mette a disposizione uno strumento molto potente: App Inventor. Vediamo di che si tratta

Prima di approfondire "App inventor" è utile, proprio per capirne l'importanza, descrivere, seppure in modo molto sommario, le procedure classiche per programmare in Android.
Un programma in Android, che utilizza il kernel (software di base che controlla l'hardware) di Linux, è il risultato dei byte-code di Java eseguiti non con JVM (Java Vitual Machine) ma con la DVM (Dalvik Virtual Machine).
Come si sa lo stesso programma scritto in Java può essere eseguito con qualsiasi sistema operativo perché interpretato dal JVM.
Nel caso di Android perché la DVM e non la JVM? Tenendo conto del fatto che uno smartphone o un tablet hanno una memoria ridotta, la DVM è ottimizzata per l'esecuzione delle applicazioni riducendo al massimo l'impiego della stessa sfuttando al massimo le caratteristiche del sistema.
Comunque è evidente che per realizzare un applicazione in Android bisogna conoscere, tra l'altro, la programmazione in Java. Programmare in Java è molto complesso. Perciò solo gli specialisti si impegnano a farlo. Ciò, di fatto, restringe il campo di coloro che sono in grado di realizzare tali applicazioni.
Ma Google, che è l'artefice del grande successo di Android, per rendere accessibile la programmazione agli appassionati, ha ideato e realizzato "App Inventor".
App inventor è un interfaccia grafica fra il programmatore e Android.
Semplificando al massimo si può asserire che il programmatore realizza una sorta di flow chart (diagramma di flusso) del programma ed App Inventor lo tramuta in applicazione. Non è proprio così, e lo vedremo, ma il concetto rende bene l'idea.
Perciò vengono evitate le lunghe liste di parentesi, punti, virgole etc. E chi programma sa bene quanto volte si va in error per uno spazio, un punto o una parentesi fuori posto. E poi i cicli errati, gli errori di digitazione e cosi via.
A questo punto, dopo tante parole, passiamo al concreto.

IL PROGRAMMA

Prima di tutto occorre verificare se nel pc, nel quale deve essere installato windows XP o superiore, risulta installato Java 6 o

Java 7. Occorre inoltre , se non l'avete già fatto, creare un account di posta elettronica con Google.
Ciò perché in realtà non lavoreremo con un programma sul nostro PC. Sarà il collegamente internet al sito http://beta.appinventor.mit.edu/#5948054, dove è residente il programma, che ci permetterà di realizzare le nostre applicazioni.
Nella videata che apparirà inserirete la vostra email e la pw. Quindi cliccare su "new" e inserite il nome del programma che volete realizzare, nel nostro caso "verifica_pw" che corrisponde al nostro programma di prova. Infine vi apparirà la videata di figura 1.
A sinistra sono rilevabili le varie "palette" (insieme di funzioni e comandi). Noi in questo esempio ci limiteremo ad utilizzare la palette Basic. Si intravedono già le molteplici applicazioni previste dal sistema in esame. Clicchiamo su Basic e analizziamo il suo contenuto riportato in Figura 2.
A questo punto, prima di procedere, analizziamo il programma che vogliamo realizzare ed installare sul nostro smartphon o tablet. Vogliamo ottenere un'applicazione che ci si presenta con un icona; cliccando su questa si dovrà aprire una videata che mostri due button (pulsanti), una label (etichetta) ed un tex box (una riga dove inserire del testo). Nel tex box inseriremo una pw. Quindi pigeremo sul button che chiameremo verifica. In basso al tasto la label risposta. Se la pw sarà corretta la label risposta mostrerà "pw corretta", altrimenti "pw errata". Il secondo button consentirà l'uscita dall'applicazione.
Inseriamo in Screen 1 tutte le funzioni e i comandi occorrenti (vedi Figura 3).
L'inserimento avviene trascinando all'interno di Screen 1 gli elementi scelti. Nella fascia components viene riportato automaticamente l'ordine e la sequenza degli elementi scelti. Si noti che quando un elemento viene introdotto compare una riga blu. Questa indicazione ci consente l'inserimento sopra o sotto dell'elemento. Per inserimenti diversi si utilizzano le funzioni di Screen Arrangement visibile in basso a sinistra. Questa opzione crea una finestra all'interno della quale si possono inserire gli elementi o affiancati o in altre posizioni.
Nel nostro caso Label 1 precede TexBox1.
A quest punto bisogna dare dei nomi alle label e ai button.

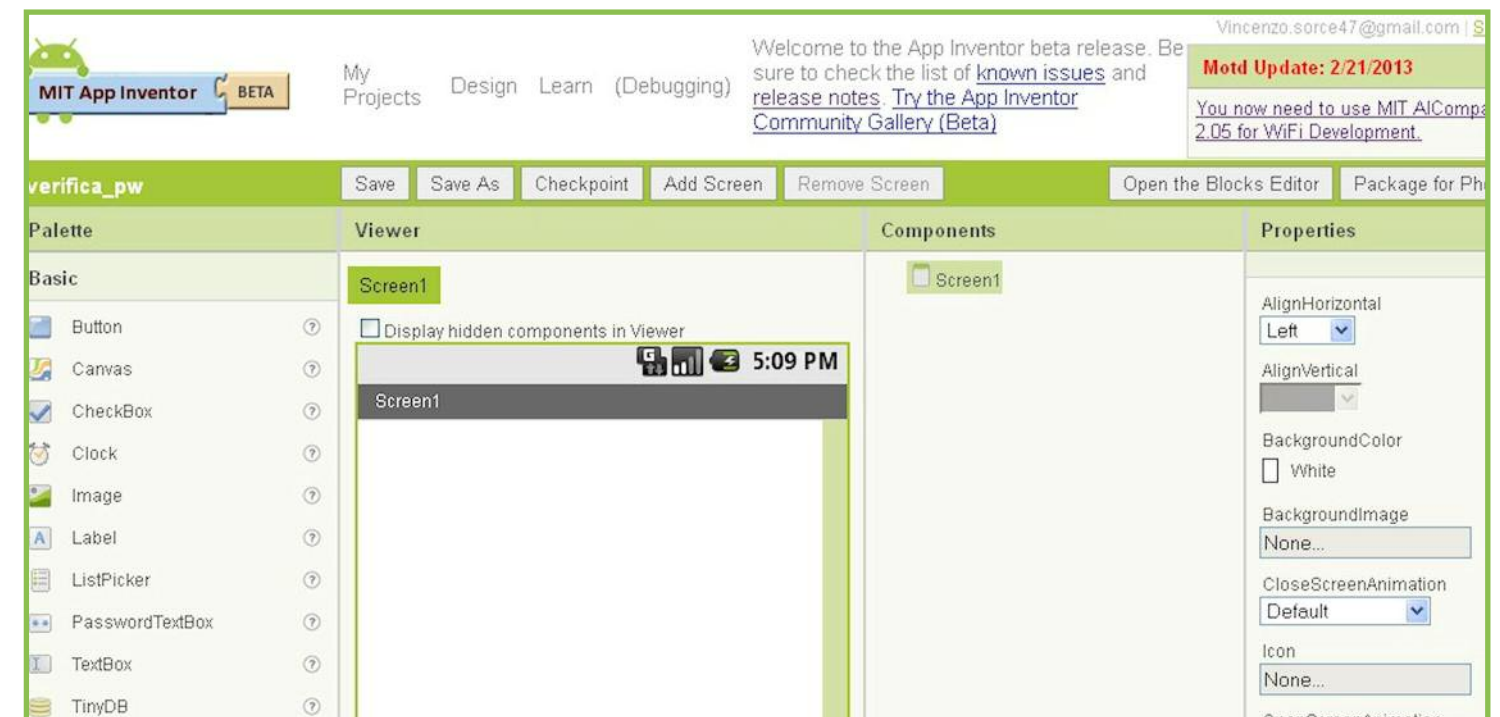



Figura 2: Comandi e funzioni della "Palette -> Basic".

Esaminiamo la Figura 4.
Selezionando in ordine i componenti inseriti in Screen 1 nella colonna Components, è possibile attribuire un text inserendolo nella colonna Properties.
Così, selezionando nella colonna Components la label 1, inseriremo in Text : "inserire la pw";

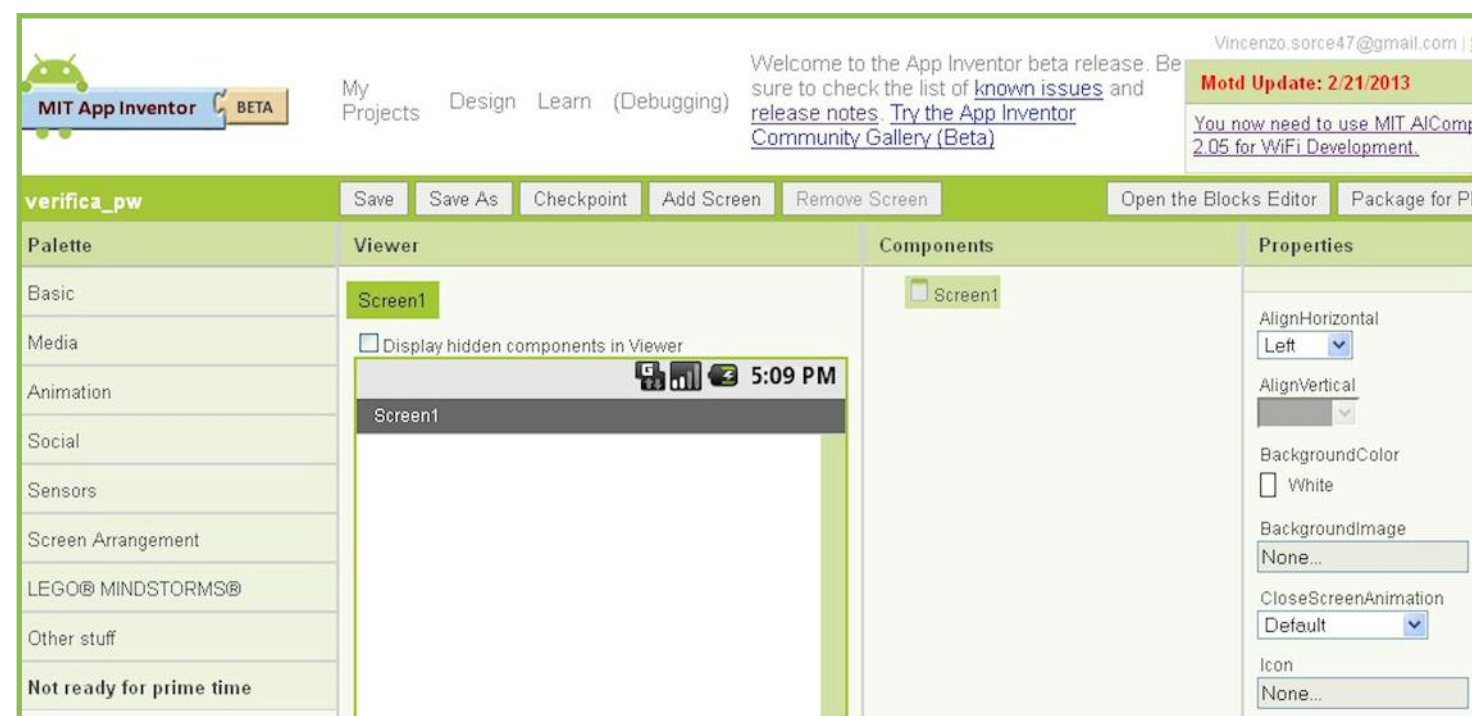



Figura 1: Videata iniziale del programma

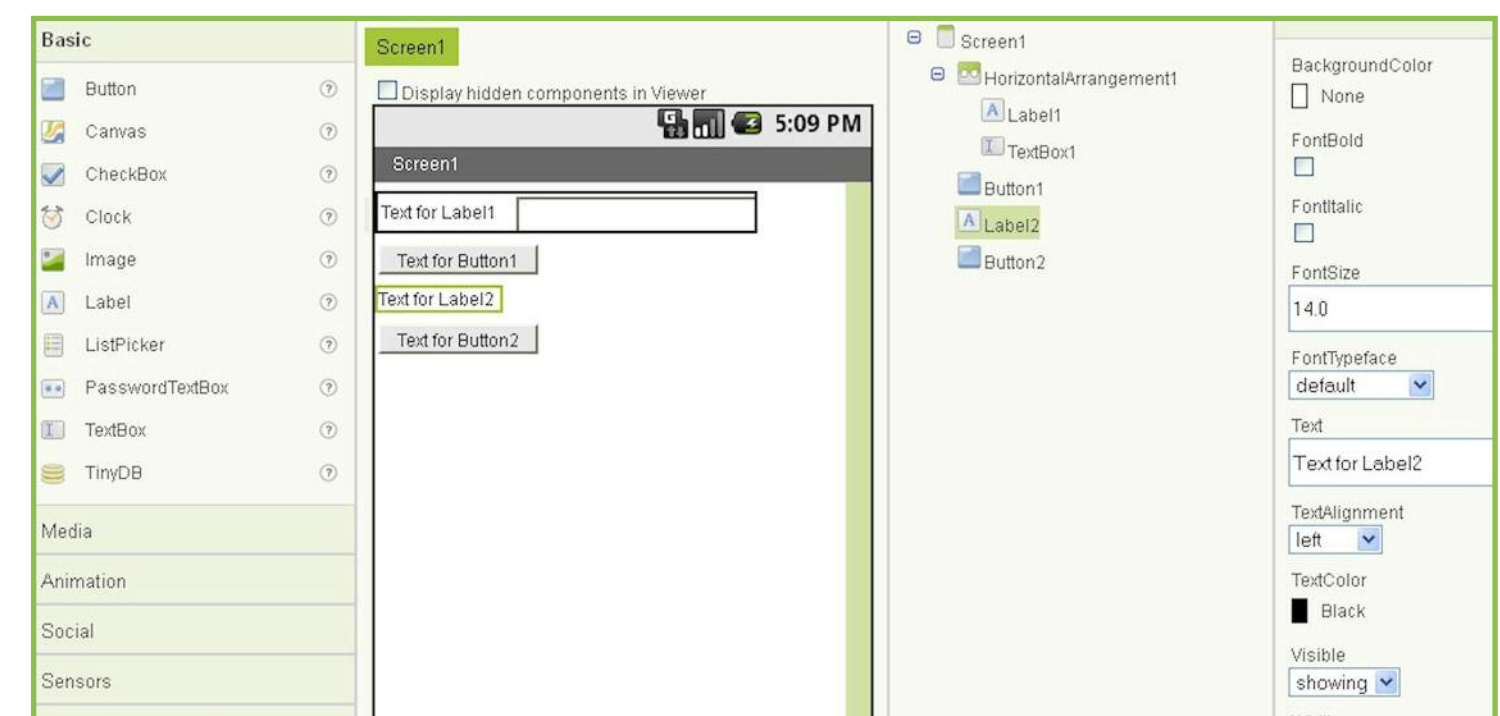



Figura 3: Inserimento dei Comandi e delle Funzioni.

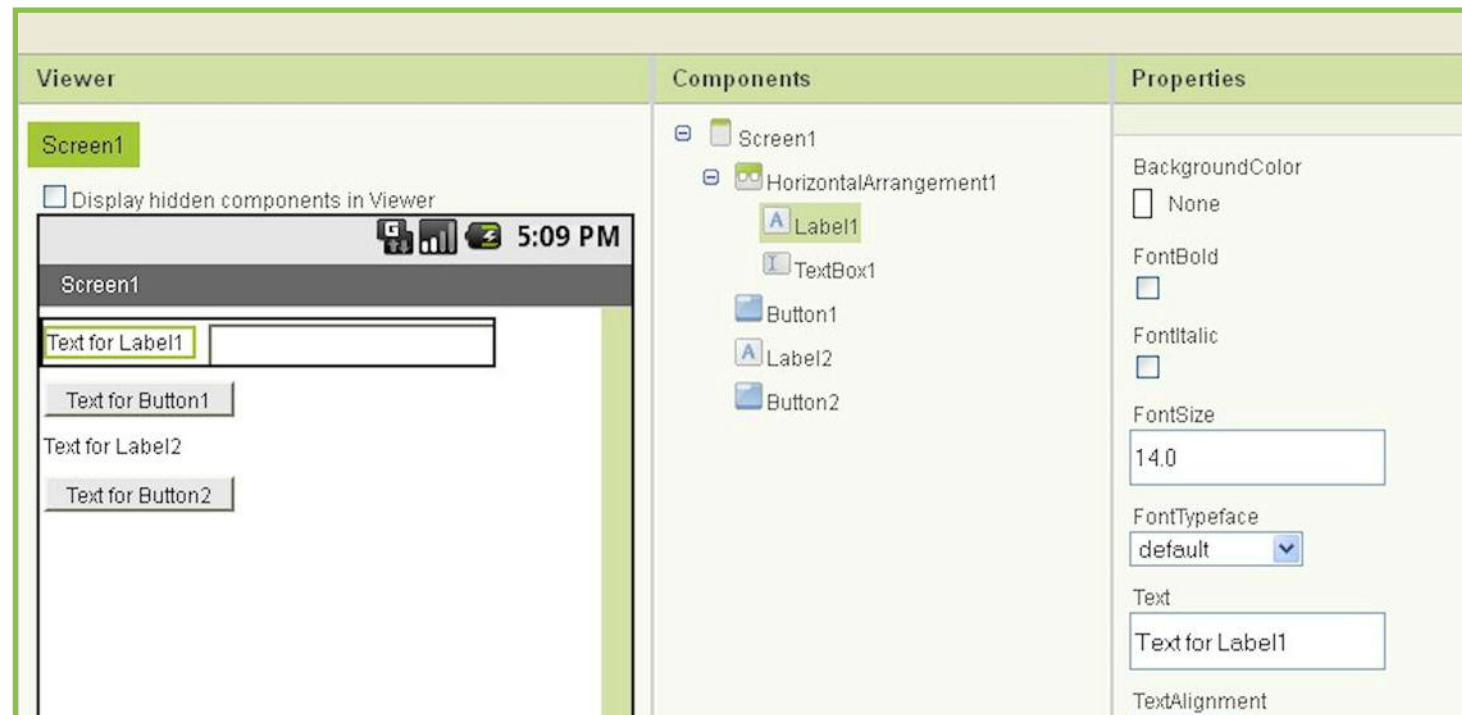



Figura 4: Attribuzione dei nomi a “Label” e “Button”.

Figura 5: Richiesta di apertura del “Codeblocks”.

Per TexBox1 cancelleremo la scritta in Hint (messaggio) dato che è il luogo dove inseriremo la pw. Analogamente si procederà con gli altri componenti.
Fin qui abbiamo stabilito quali sono i componenti da utilizzare nel nostro programma ed il loro testo. Adesso dobbiamo stabilire le loro azioni.
Clicchiamo su Open the Blocks Editor.
Poiché il comando sarà trasmesso via internet, può accadere che il risultato apparirà nello schermo dopo qualche secondo (dipende dalla velocità del collegamento). Appariranno nell’ordine le schermate 5, 6 e 7. Nella videata mostrata nella Figura 7 stabiliremo le relazioni fra i componenti ed infine otterremo la nostra applicazione da trasferire ed avviare nello smartphone o ne tablet.

Figura 6: Avviso del Download dell’applicazione.

Esaminando quest’ultima si nota al centro delle tre colonne a sinistra l’elenco dei nostri componenti. Clicchiamo su Button 1: saranno visualizzati una serie di box con condizioni diverse. Scegliamo il primo trascinandolo al centro delle videata. Abbiamo scelto questo box perché soddisfa la condione when button1.click then do che vuol dire: quando premo il bottone 1 fai…
Adesso dobbiamo stabilire cosa fare. Allora passiamo nella colonna Built-in (costruisci dentro), per stabilire cosa deve eseguire il nostro programma. Selezionamo Control e selezioniamo il box che ci permettono la condizione if then else. Per ora non la inseriamo dentro il Button1. Selezionamo Logic e scegliamo il box che ci consenta il raffronto fra la pw inserita e quella stabilita. Infine selezioniamo TexBox inserito e il tex corrispondente alla pw stabilita. Tutto ciò è riscontrabile nella Figura 8.
A questo punto incastriamo i vari box ed otterremo quanto riscontrabile nella Figura 9 (sono stati inseriti anche i rimanenti componenti).
Sembrerebbe tutto facile. Non è così. Si intuisce subito che è andato tutto a posto perché abbiamo fatto le scelte giuste e gli incastri coincidono. A proposito, quando l’incastro è giusto il programma emette un rumore che ne conferma la correttezza dell’inserzione.
Per salvare il nostro programma basta cliccare su Saved in alto a sinistra. Nel caso in cui tale scritta non compare in grassetto vuol dire che è stato salvato automaticamente.
Per scaricare l’applicazione sul PC bisogna tornare sulla videata iniziale. Bisogna ricordare che tale videata deve essere

Figura 7: Display del “Codeblocks”.

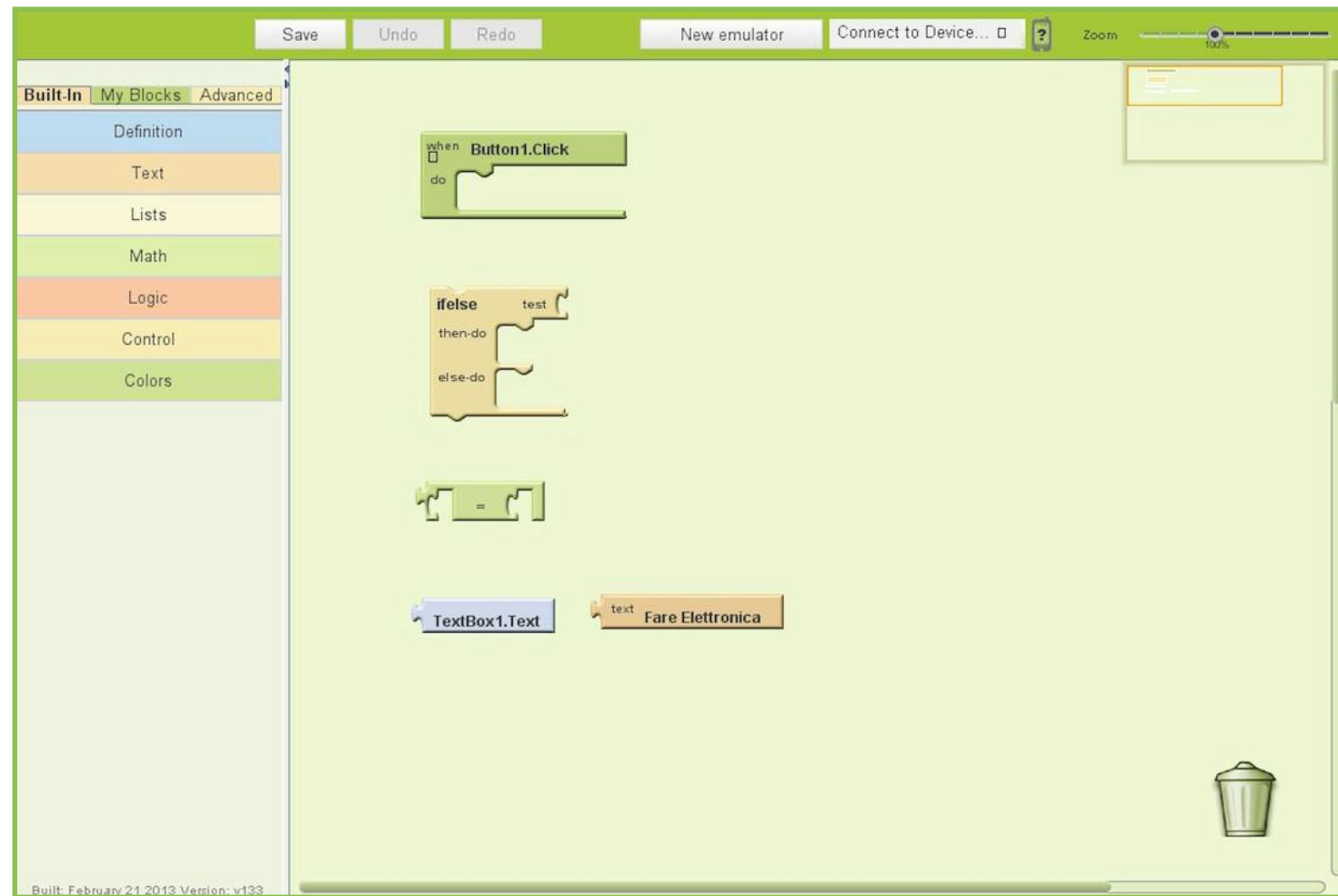



Figura 8: Scelta dei Comandi e delle Funzioni.

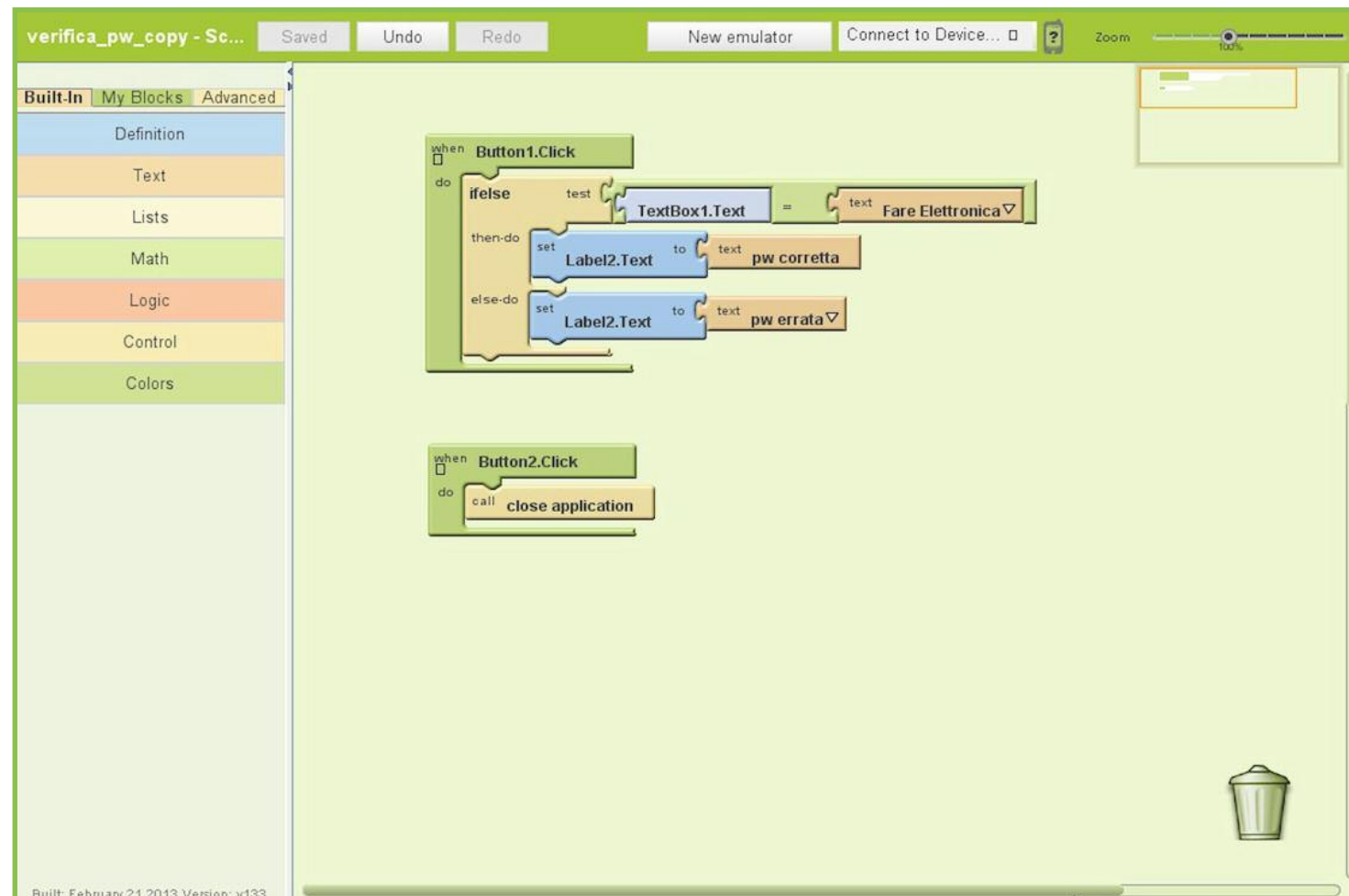



Figura 9: Assemblamento dei Comandi e delle Funzioni.

chiusa soltanto dopo aver scaricato sul PC l'applicazione.
Clicchiamo su Package for Phone e scegliamo Download to this computer. Ci troveremo l'applicazione la dove scegliamo di scaricare i file. Dovremo adesso copiare l'applicazione sul tablet o smartphone. Il modo più scontato è quello di copiarlo sulla micro SD estraibile. Si può, se si ha a disposizione la porta USB, trasferire il file direttamente.
Per installare l'applicazione basta cliccare sulla stessa. Apparirà la schermata di figura 10. E' riscontrabile in alto a destra della Figura 9 un rettangolo grande con, all'interno, un rettangolo piccolo. Questa opzione consente, quando un programma è molto esteso, di scorrerlo agevolmente.

Figura 11: Apertura dell'applicazione dopo aver cliccato sull'icona.

Un'altra soluzione, per rendere ill programma più schematico, è quello di cliccare sul quadratino in alto a sinistra di ciascun box. In tal caso il box stesso si riduce ad un rettangolo con la scritta principale.

di GIORGIO OBER

INTERFACCIAMENTO DEI PROCESSORI
LA LETTURA DEL TASTIERINO

(parte 17ª)

Dopo aver approfondito le possibili interfacce per le unità di visualizzazione, a Digit o a Matrici di LED, rimangono da vedere quelle per un altro componente, molto utile nel progetto di strutture basate sui microcontrollori: le piccole batterie di tasti note come keypad

Nelle prime puntate di questa lunga serie di articoli sull'interfacciamento delle periferiche di I/O ci siamo già occupati della necessità di coinvolgere le porte d'ingresso di un microcontrollore o le linee di input di una porta parallela nella lettura dello stato di una batteria di contatti di vario genere, associati ai più disparati dispositivi d'ingresso, come gli switch di finecorsa, o i contatti reed di un impianto di antifurto, o i sensori di livello di un controller per liquidi, oppure tastiere o deviatori, e così via.
Ma nella realizzazione di apparecchiature elettroniche controllate da microprocessori (in tutti i settori come quello industriale civile o militare, alimentare, medico, dei giochi o dei trasporti, ecc..) è piuttosto frequente la necessità di coinvolgere piccole tastiere pronte per l'uso.
Esse, note con il nome inglese di Keypad, sono di aspetto rettangolare e sono dotate di 4, 8, 12 o 16 tasti organizzati a Matrice Riga x Colonna; il loro utilizzo si è diffuso rapidamente anche per il prezzo piuttosto contenuto (dovuto alle moderne tecnologie a basso costo) e per il grado di eleganza e professionalità che conferisce ai nostri progetti.
La versione a 12 tasti è la tipica tastiera telefonica che porta, oltre alle 10 cifre del sistema di numerazione decimale, anche i simboli "*" e "#"; ma se il circuito programmabile è chiamato ad interagire direttamente con la CPU può essere utile disporre di tastierini da 16 tasti, uno per ciascuno dei simboli del sistema di numerazione esadecimale (i numeri da 0 a 9 e le lettere da A a F), per esempio per comporre l'indirizzo di una locazione di memoria o per predisporre il valore binario da inserire in un registro o in una cella di RAM.
Prima di affrontare l'analisi della loro struttura e delle rispettive necessità d'interfacciamento può essere interessante studiare le problematiche relative alla gestione di tastiere esteticamente simili ma realizzate con normali pulsanti, per i quali sarà necessario predisporre un adeguato circuito stampato o una congruente sistemazione su piastra millefori.
La scelta migliore è quella di affidarci a dispositivi aperti a riposo (NA, normalmente aperti) in grado di forzare sugli ingressi un livello 0 (massa) quando sono chiamati ad esercitare la loro azione; la **Figura 1** si riferisce alla gestione di 16 pulsanti, direttamente collegati a due porte (PORTA e PORTB) di un normale microcontrollore a 8 bit; ciascuna delle loro linee è programmata in ingresso e, nella situazione di attesa, è trovata a 1 logico, cioè (come tipicamente succede per ogni ingresso TTL) si porta allo stato alto e lo mantiene.
Poiché ciascuna linea d'ingresso (se lasciata libera, "fluttuante", cioè non collegata a nulla) può portare a commutazioni indesiderate (in presenza di un qualunque rumore accoppiato capacitivamente ad essa, a causa della sua capacità parassita) è caldamente consigliata la presenza dei resistori di pull-up, per collegarla a 1 anche fisicamente; il valore di resistenza utilizzato è tipicamente dell'ordine del Kohm per imporre una caduta ai suoi capi compatibile con il livello logico minimo da assicurare (V_{IH}, pari a circa 2V) in presenza della pur piccola la corrente I_{IH} assorbita in

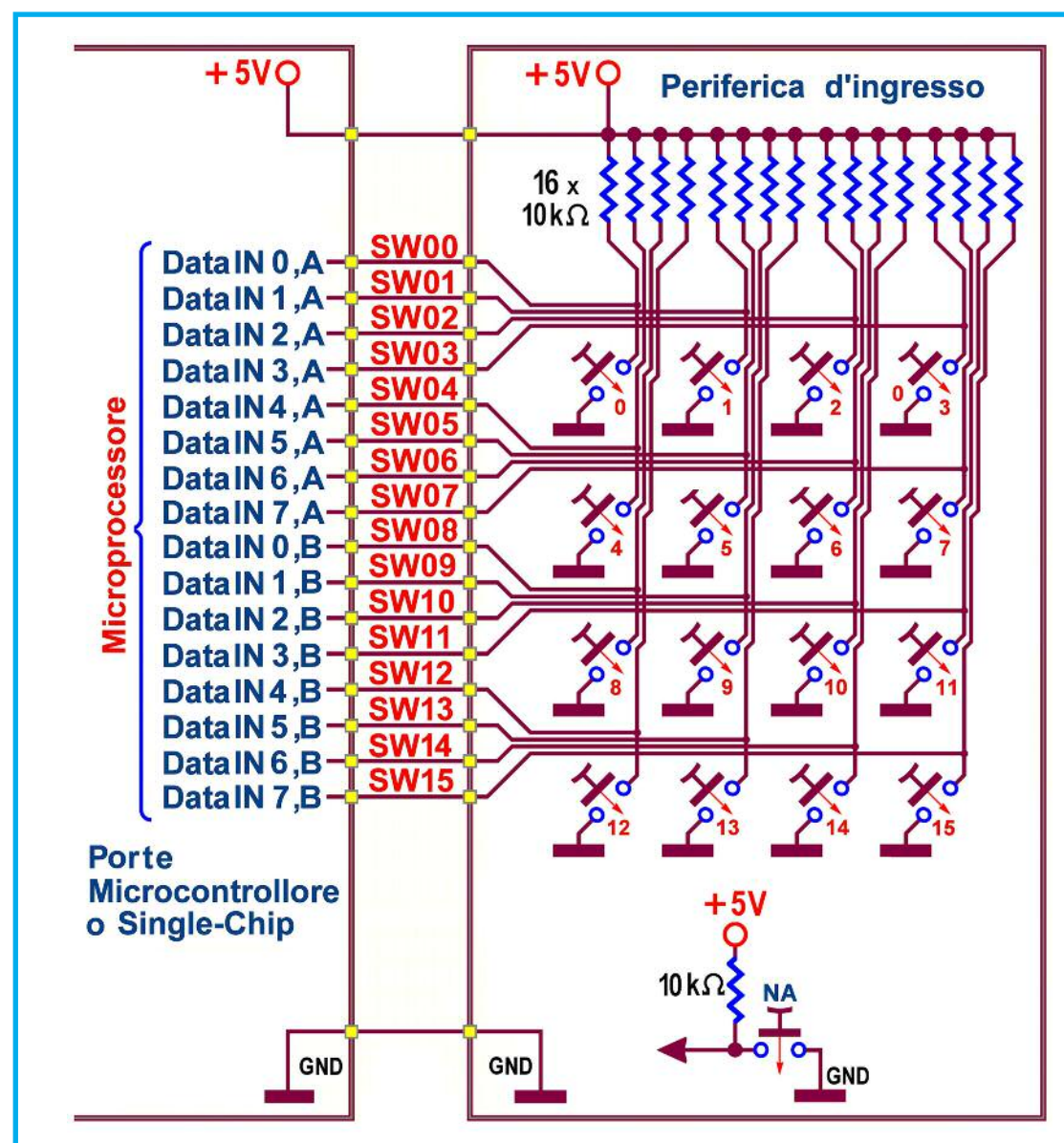



Figura 1 - Lettura di 16 pulsanti operata direttamente dalle porte PORTA e PORTB di un microcontrollore

ingresso a livello alto (al massimo 20 A per la TTL LS).
Va detto che molti microcontrollori garantiscono già la presenza di un resistore di pull-up interno su ciascuna linea programmata in ingresso; in questo caso (ma è bene esserne sicuri) la presenza dei 16 resistori da 10 kOhm può essere evitata.
Il tipico programma che gestisce questa soluzione rimane in attesa (in polling) eseguendo (con semplici istruzioni all'interno di un ciclo ripetitivo, loop) la lettura sincronizzata, uno dopo l'altro, dei 2 bytes disponibili in ingresso, di fatto offrendo continuamente al processore interno la parola FFFFH (uguale, in binario, a 16 1 di fila) fino a quando la pressione di uno o più pulsanti forza a 0 il bit associato alla linea coinvolta; di solito, in questo preciso istante, il codice operativo smette di interrogare la batteria di contatti (esce dal loop) e attiva la procedura di servizio del primo sentito premuto; oppure attiva una ricerca mirata (solitamente con una maschera binaria) nei confronti di un determinato tasto, cercando di verificare se esso è stato posto a massa.
Non va poi dimenticato che l'azione attiva bassa è esercitata da contatti meccanici: la loro chiusura e apertura (in seguito alla pressione o al rilascio di un tasto) provoca un fastidioso problema legato al fatto che, all'interno di questo dispositivo, una barretta o una piccola lamina di metallo conduttore viene spostata (con una certa energia) da una posizione ad un'altra, posta poco distante.
Per la sua natura intrinsecamente elastica questo elemento meccanico interno "vibra" in prossimità del pin d'arrivo, provocando alcuni inevitabili micro-rimbalzi che, per qualche decina di millisecondi, non gli permettono di assestarsi immobile su di esso, al fine di garantire un contatto finale stabile; questo fenomeno, noto come key bouncing, dovrà essere affrontato e risolto in ciascuno dei progetti proposti in questo articolo.
In verità esso affligge tutte le tastiere, anche quelle in plastica e in gomma, nelle quali il contatto è ottenuto abbassando una minuscola cupola in gomma su un piccolo pettine di piste, impresse informa circolare sul circuito stampato.
La quantità, la durata e l'ampiezza dei singoli impulsi spuri è imprevedibile: passando attraverso la porta d'ingresso essi subiscono una "squadratura", per effetto del livello raggiunto in ingresso da ciascuno di essi, riferito ai valori tipici V_{IH} (2V, minimo ritenuto 1) e V_{IL} (0,8V, massimo ritenuto 0); data la velocità di esecuzione delle istruzioni del processore (mille volte più rapida degli impulsi spuri) il programma di controllo non potrà fare a meno di rilevarli.
Senza provvedere ad eliminare il problema, il servizio di un tasto premuto potrebbe partire più volte, con effetti devastanti sul corretto funzionamento di una determinata applicazione; si pensi a quella di un contatore dotato di un'unità di visualizzazione: se si incrementasse il suo conteggio in funzione della lettura non protetta di un pulsante, il numero sul digit sarebbe certamente inaffidabile, aumentando (in modo assolutamente imprevedibile) di due o più unità, invece di incrementarsi.
Per questo un buon programma di gestione, dopo aver rilevato la pressione di un tasto, deve attivare una procedura di ritardo (di una o due decine di ms) e poi tornare a leggere il suo stato, prima di far partire il servizio per esso previsto. Questa tecnica di programmazione evita di far intervenire il controllo in presenza di falsi contatti, operando un key debouncing (antirimbalzo) da software.
La soluzione diretta, suggerita dalla **Figura 1**, è decisamente improponibile, perché sottrae al single-chip quasi tutte le risorse di I/O disponibili: bisogna quindi pensare ad un strato di hardware in grado di minimizzarle, pur garantendo lo stesso servizio; di solito la presenza di un'interfaccia aumenta i costi del progetto e complica (sia pur lievemente) il software di gestione.
La **Figura 2** è presenta una prima soluzione che organizza la lettura dei 16 tasti suddivisi in 4 quaterne, assunte dalle 2 metà di due 3-State Octal Buffers/Line Drivers, le uscite (non invertenti) dei quali sono allacciate tra di loro per formare un bus monodirezionale e proposte sulle 8 linee dalla PORTB, tutte programmate in ingresso; è necessaria anche una linea programmata in uscita, supportata dal bit0 di PORTA, il cui valore logico va mantenendolo stabile per tutto il tempo necessario per consentire al programma di leggere la prima metà dei tasti (SW00-SW07, se posta a 0) o la seconda metà (SW08-SW15, se posta a 1).
Sono possibili due varianti: a) lo schema

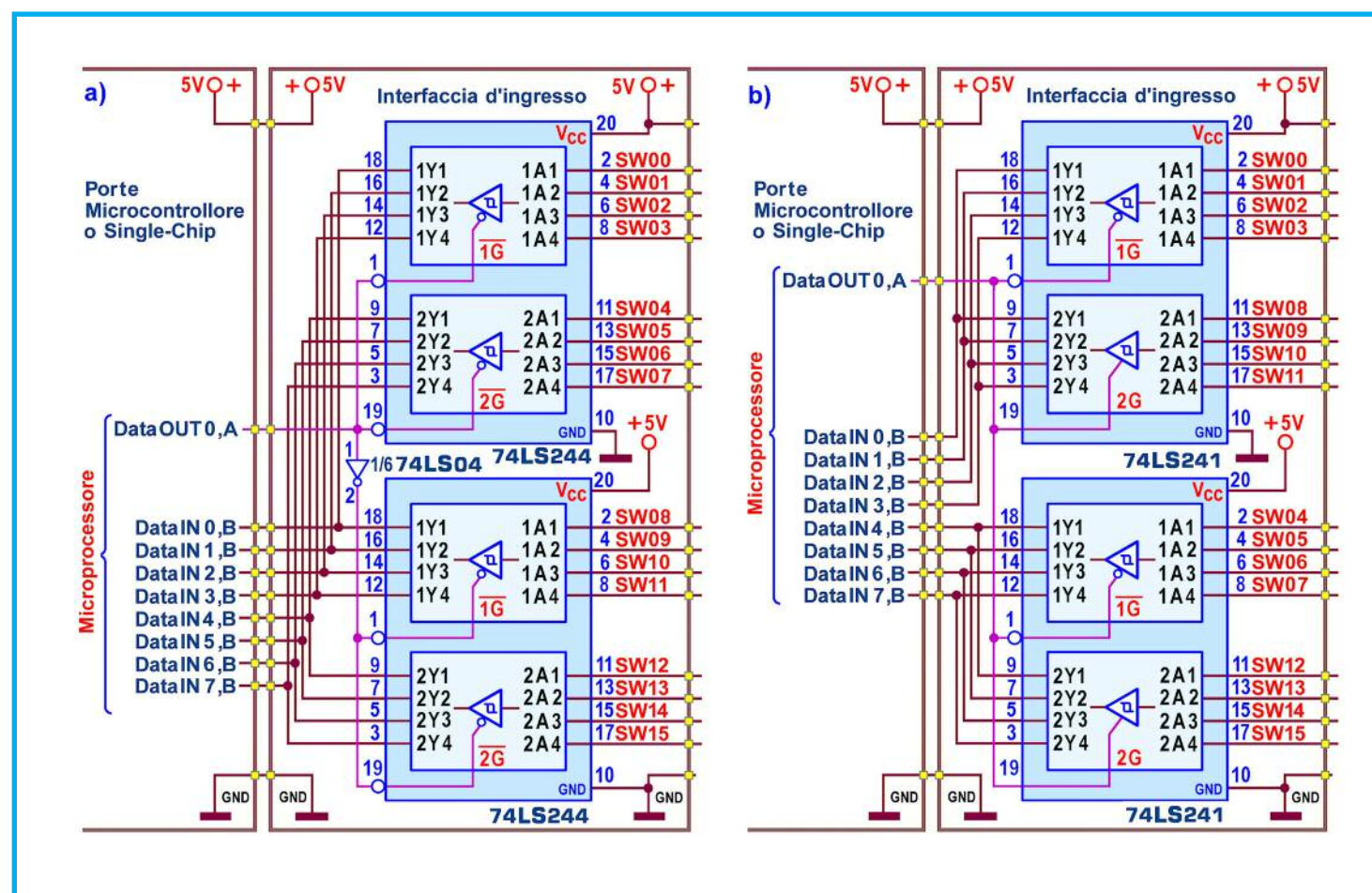



Figura 2 - Lettura di 16 pulsanti con 3-State Octal Buffers/Line Drivers 74LS241 e/o 74LS244 via microcontrollore

Figura 3 - Lettura di 16 pulsanti con un 16-Line To 1-Line MUX 74150 via microcontrollore

utilizza due 74LS244 in parallelo: la sua linearità nel servire i 16 contatti è pagata con la necessità di usare un terzo integrato (un 74LS04 contenente 6 inverter) per controllare le linee di Out Enable delle 4 quadruple di buffers, tutte attive basse; b) la presenza dell'inverter può essere evitata utilizzando due 74LS241, in virtù del fatto di disporre di linee di abilitazione (1G e 2G) attive su livelli opposti, così che il valore del bit0 di PORTA sarà ora attivo solo sulla metà alta o solo su quella bassa di entrambi i componenti; questa opportunità si traduce nel bisogno di porre maggior attenzione nel collegamento sugli ingressi dei 16 tasti, al fine di garantire la corretta codifica dell'informazione acquisita dalla PORTB.

La seconda soluzione affida la lettura dei 16 tasti ad un Data Selector/Multiplexer (o MUX) 74150, un dispositivo chiamato a trasferire l'informazione presente sui suoi 16 ingressi (solo uno alla volta) verso l'unica uscita prevista; il suo funzionamento ricorda quello di un commutatore rotativo "ad una via, 16 posizioni", un componente meccanico molto utilizzato nella progettazione elettronica, rispetto al quale offre il vantaggio di un controllo digitale decisamente versatile, nel senso di poter puntare un qualunque ingresso senza dover rispettare la stretta sequenza imposta dalla rotazione oraria o antioraria del perno meccanico.

Lo schema di **Figura 3** mostra i collegamenti necessari: la selezione degli ingressi è operata dal codice binario imposto sugli 4 ingressi Data Select, affidato a 4 delle linee dalla PORTA, programmate in uscita, avendone immediatamente a disposizione lo stato logico complementato di ciascuno, sull'uscita W, collegata alla linea 0 della PORTB, programmata in ingresso.

Il software di gestione può sottoporre a scansione consecutiva tutti i 16 tasti, proponendo la sequenza da 0000 a 1111 sui primi 4 bit della PORTA e verificando (dopo l'emissione di ogni nibble) lo stato del bit0 di PORTB; ma può anche puntare e testare un singolo pulsante, mantenendo fisso il suo codice a 4 bit su PORTA; in ogni caso il relativo pulsante verrà ritenuto premuto se il bit0 viene trovato a 1, a causa dell'inversione logica imposta internamente all'uscita W.

Da notare che questo dispositivo svolge il suo compito solo se l'ingresso G (Strobe) è collegato a massa: quando esso non è attivo (cioè a 1 o scollegato) l'uscita del MUX è forzata a 1, qualunque sia il valore di Data Select, rendendola inutilizzabile.

Una soluzione originale e sofisticata è quella offerta in **Figura 4**, che utilizza un paio di Priority Encoder 74LS148; la presenza di un livello logico attivo su una qualunque delle sue 8 linee d'ingresso è rilevata e codificata sulle sue 3 linee d'uscita con una parola binaria da 000 a 111.

La particolare struttura interna di questo dispositivo rende possibile la codifica di un solo ingresso, anche se ne risultano attivi 2 o più contemporaneamente: essa assicura infatti una logica di priorità in grado di garantire la codifica solo all'ingresso di peso maggiore: tutti gli altri, eventualmente attivi con priorità inferiore, saranno ignorati.
Sebbene poco pratico per una tastiera convenzionale, questo meccanismo è decisamente interessante se, come nel caso di una keypad, si da per scontata la pressione di un solo tasto alla volta.
Il funzionamento del 74LS148 è consentito solo se il segnale Enable Input (EI, attivo basso) è forzato a massa: in caso contrario l'encoder porta alte tutte e 5 le uscite; la sua Tabella di Verità (vedi **Figura 5**) aiuta a comprendere ogni dettaglio.
Con EI=0, se nessuna delle linee d'ingresso risulta attiva, le uscite Enable Output e Group Signal (EO e GS, entrambe attive basse) sono poste rispettivamente a 0 e a 1, e le 3 Data Output (pure attive basse) sono, esse pure, tutte a 1.
Non appena una o più delle 8 linee d'ingresso (attive basse) viene fissata a 0 .. GS passa a 0 e EO e passa a 1; immediatamente sulle 3 uscite Y2, Y1 e Y0, viene scritto il codice binario a tre bit che esprime in complemento a 1 il numero d'ordine dell'unico pulsante premuto o di quello (in caso di pressione multipla) che ha la priorità massima.

INGRESSI									USCITE				
I0	I1	I2	I3	I4	I5	I6	I7	EI	EO	GS	Y2	Y1	Y0
X	X	X	X	X	X	X	X	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0	1	1	1	1
X	X	X	X	X	X	X	0	0	1	0	0	0	0
X	X	X	X	X	X	0	1	0	1	0	0	0	1
X	X	X	X	X	0	1	1	0	1	0	0	1	0
X	X	X	X	0	1	1	1	0	1	0	0	1	1
X	X	X	0	1	1	1	1	0	1	0	1	0	0
X	X	0	1	1	1	1	1	0	1	0	1	0	1
X	0	1	1	1	1	1	1	0	1	0	1	1	0
0	1	1	1	1	1	1	1	0	1	0	1	1	1

Figura 5 - 8-to-3 Line Priority Encoder 74LS148: Tabella di verità

Per chiarezza, la priorità massima è affidata all'ingresso I7 e, quando I7=0 (indipendentemente dal valore delle altre 7 linee, che potrebbero essere anch'esse tutte a 0) il valore sulle uscite è Y2Y1Y0=000 (cioè il numero 7, in binario 111, complementato); la stessa logica è applicata alle altre possibili situazioni, fino al caso in cui solo I0 è a 0 (con tutte gli altri 7 ingressi a 1), che lascerà le uscite a Y2Y1Y0=111.
Risulta evidente che l'uscita EO si presta in modo ottimale per assicurare l'espandibilità di questa tecnica; il progetto di **Figura 4**, mostra come i due 74LS148 debbano essere posti in cascata; quello in basso si occupa degli ingressi più pesanti e, solo se nessuno di essi è attivo, concede a quello in alto il permesso di funzionare: solo in questa situazione, infatti, la sua linea EO è a 0, forzando a 0 anche al piedino EI di quello a valle (cioè abilitandolo all'interpretazione degli ingressi con priorità inferiore).
Anche l'uscita GS è molto importante: quando vale 0 significa che almeno un tasto risulta premuto; per monitorare tutti e 16 i pulsanti basterà interrogare l'uscita dalla AND che raccoglie i Group Signal di entrambi gli integrati, collegata alla linea 4 della PORTB, programmata in ingresso come le altre (dalla 0 alla 3), sulle quali sarà disponibile la codifica associata al tasto premuto con priorità, cioè 0000 per SW15, ... , 1111 per SW00.
Naturalmente il segnale GS in uscita dalla AND può essere utilizzato anche come INT (interrupt) rendendo immediata l'operazione di servizio del pulsante e liberando una linea di I/O: l'esercizio dell'interruzione della CPU è una prassi forse un po' complicata ma che consente risultati certamente più efficienti di quelli ottenuti tramite polling.
Nelle considerazioni precedenti abbiamo verificato che, per il servizio di un tastierino da 16 pulsanti, sono necessari 17 fili, se la logica che li controlla dispone di resistori di pull-up interni, oppure 18, se è necessario assicurare anche la tensione positiva per governare la loro presenza; l'impiego di un'interfaccia ci ha consentito una drastica riduzione del numero di linee necessarie, naturalmente in cambio del costo dei componenti coinvolti.
Ma esiste un'altra filosofia d'impiego, già vista in passato per la gestione di grandi quantità di LED organizzati per mostrare numeri o caratteri: la connessione "a matrice", che consiste nell'organizzare i contatti in Righe e Colonne, posizionandoli in prossimità dei nodi d'incontro della struttura; con tastiere da 16 tasti sono così necessarie solo 8 linee, ridotte a 7 con 12 tasti.
Queste piccole tastiere (note come Keypad) sono già disponibili su piccoli sup-

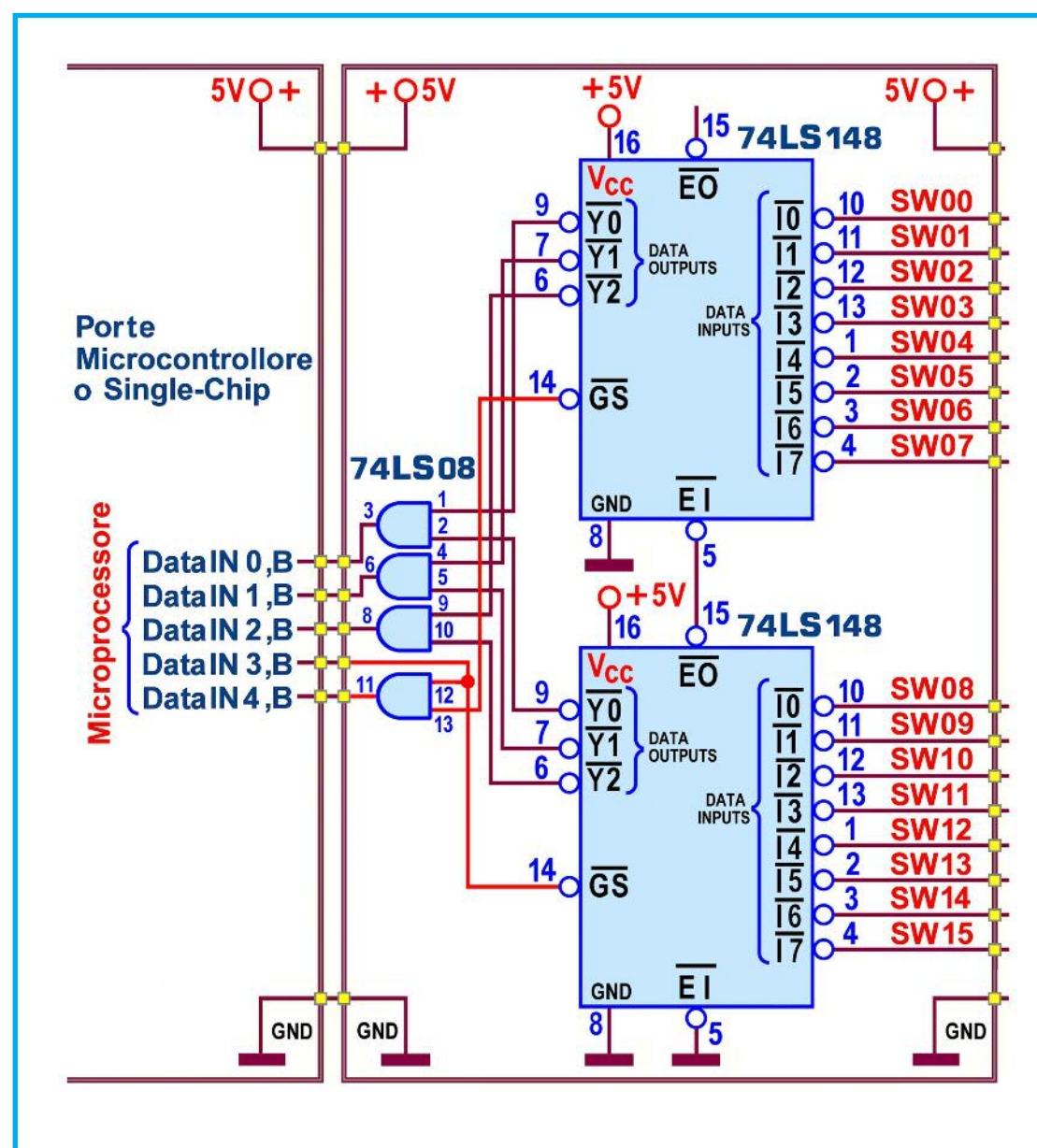



Figura 4 - Lettura di 16 pulsanti con Priority Encoder 74LS148 via microcontrollore

Keypad	Output Pins
1	3 - 2
2	1 - 2
3	5 - 2
4	3 - 7
5	1 - 7
6	5 - 7
7	3 - 6
8	1 - 6
9	5 - 6
0	1 - 4
#	5 - 4
*	3 - 4

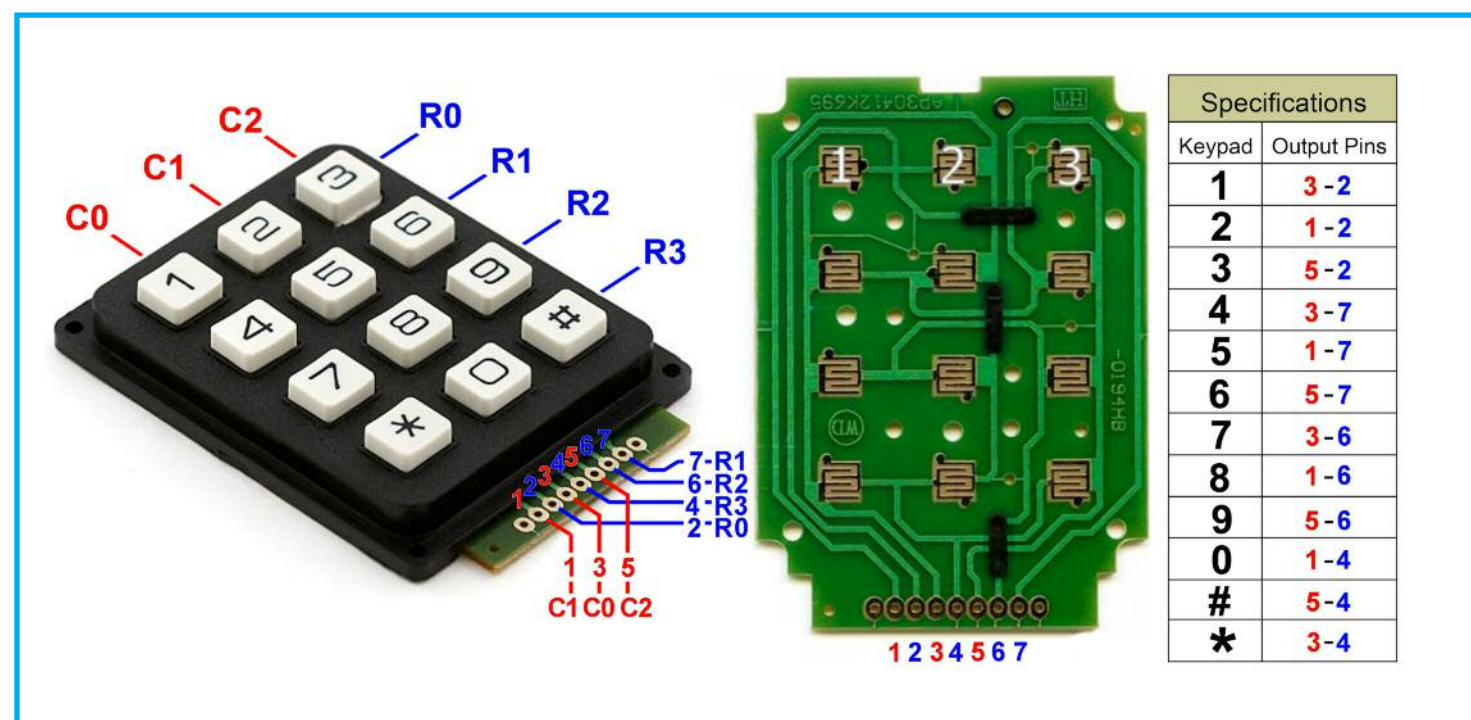



Figura 6 - 12 Button Keypad COM-08653, prodotta dalla SparkFun Electronics

porti in vetronite, in forma compatta e funzionale, dotati di un connettore di riferimento che rende facile la localizzazione e la lettura dei singoli contatti; la **Figura 6** mostra una tipica versione, la Keypad COM-08653 a 12 tasti, prodotta dalla SparkFun Electronics.

Nella tabella allegata sono indicate le coordinate Colonna-Riga per ciascun tasto, riferite alla stringa di 9 piazzole a saldare ben visibile in basso, adatte per il collegamento di un connettore a pettine maschio o di un cavo piatto; da notare che le due piazzole esterne non sono collegate e che la sequenza delle Righe e Colonne associate alle altre 7 non è consecutiva, ma conforme alla ottimizzazione delle piste sul circuito stampato.

Il contatto fisico non è sempre di natura metallica: molto frequentemente sotto ogni tasto c'è un cerchietto di membrana conduttrice che, in seguito alla pressione, provoca il corto circuito delle sottostanti piste del circuito stampato, create ad arte nel punto di contatto per facilitare questo compito; nei dispositivi più pregiati sono realizzate con piste dorate, mentre in quelli più comuni sono stampate con una vernice che conduce l'elettricità, ben visibile (colorata di nero) anche nei dettagli di **Figura 6**.

Nonostante l'apparente fragilità dei materiali coinvolti, la durata garantita "per pulsante" è di ben 3 milioni di operazioni.

Dunque, con riferimento ad una Keypad a 16 tasti, la soluzione più logica sembra quella di affidarla al controllo diretto delle 8 linee di una porta di microcontrollore, metà programmate in ingresso e metà in uscita; la **Figura 7** mostra lo schema, con le Righe dalla Matrice collegate alle prime 4 linee della PORTA e con le Colonne connesse alle prime 4 della PORTB.

Le Righe della Matrice devono essere attivate una alla volta, forzando a massa una sola di esse e lasciando le altre 3 a 1 logico;

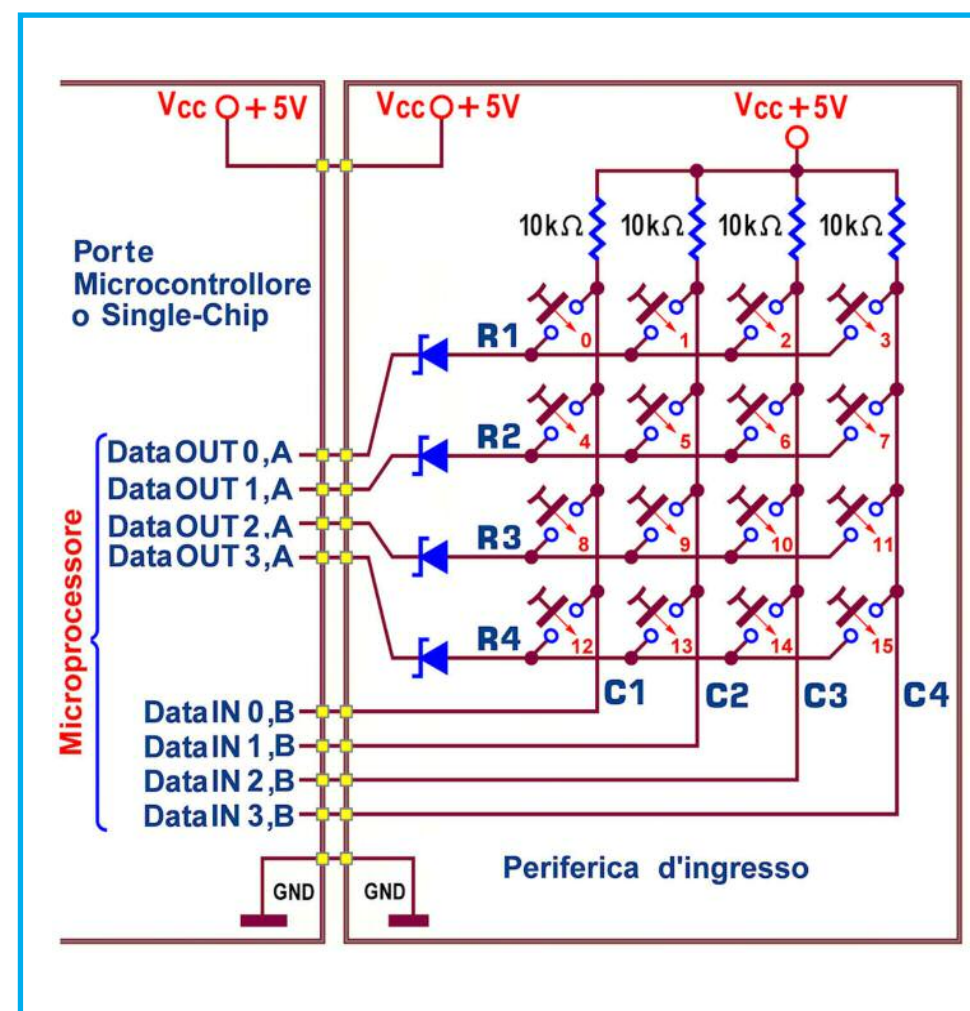



Figura 7 - Lettura di una 16 Button Keypad operata direttamente dalle porte PORTA e PORTB di un microcontrollore

dopo questa operazione deve essere eseguita la lettura delle Colonne per accertarci della presenza di eventuali tasti premuti: in condizioni di riposo sui 4 bit meno significativi della PORTB verrà letto il valore 1111 (assicurato dai resistori di pull-up, che collegano ogni Colonna al positivo dell'alimentazione), ma se viene premuto uno o più pulsanti collegati alla Riga posta a massa, anche il valore della Colonna ad essi corrispondente sarà trovato a 0.

Il programma di gestione deve dunque assicurare questa semplice sequenza di eventi: a) attiva la sola Riga R1, ponendo il nibble 1110 in uscita sui 4 bit meno significativi di PORTA; b) assume lo stato dei tasti 0, 1, 2 e 3, leggendo sui 4 bit meno significativi dalla PORTB il valore corrente delle 4 Colonne: esso sarà 0 se il tasto è premuto e 1 in caso contrario; c) aspetta un tempo di almeno 20 ms e provvede ad una seconda lettura degli stessi bit; l'operazione è necessaria per eliminare gli effetti del key bouncing, da mettere sempre in conto nella gestione delle tastiere; d) confronta i valori ottenuti con quelli della lettura precedente, facendo poi partire il servizio solo per i tasti trovati ancora a 0; e) ripete le operazioni descritte nei punti precedenti per le altre tre righe ponendo sui 4 bit meno significativi di PORTA, rispettivamente, i valori 1101 per R2, 1011 per R3 e 0111 per R4.

Da notare la presenza dei diodi, in serie alle linee d'uscita della porta PORTA: la loro presenza è spesso disattesa, ma il rischio che si corre, in loro assenza, è piuttosto serio: sebbene l'utilizzo di questa periferi-

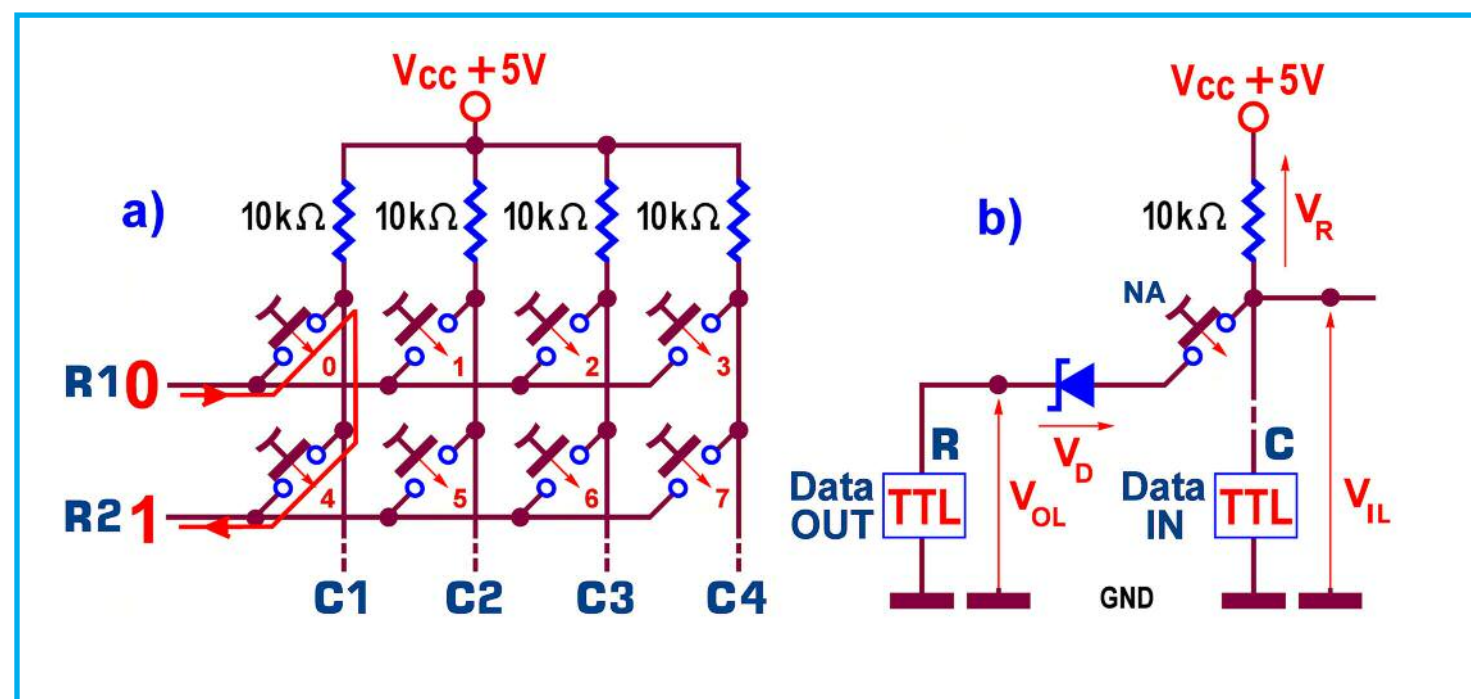



Figura 8 - La presenza del diodo, a protezione delle linee d'uscita di PORTA

ca richieda, di norma, la pressione di un solo tasto alla volta e sebbene la collocazione dei suoi tasti, ben distanti tra loro, renda piuttosto remota la probabilità di premerne più di uno, questo evento (per sbadataggine o per azione maldestra) può sempre succedere.
Se si tratta di quelli collegati alla stessa Riga non succede nulla di irreparabile, ma se si premono due (o più) tasti sulla stessa Colonna una delle uscite della PORTA può subire danni irreparabili; la **Figura 8a** aiuta a capire cosa succede: la desiderata pressione del pulsante "0" trasferisce regolarmente sulla Colonna C1 lo 0 logico predisposto sulla Riga R1, mantenendolo il tempo necessario per essere letto sul bit0 di PORTB; ma, in assenza dei diodi di protezione, la pressione simultanea del pulsante "4" forza lo 0 anche sull'uscita TTL associata alla Riga 2, posta da programma a livello alto; è facile capire che su essa si verifica un cortocircuito che la può danneggiare irrimediabilmente!

Poiché il costo di 4 diodi è sostanzialmente irrisorio il loro mancato utilizzo sembra irresponsabile; nel merito, possiamo fare qualche altra considerazione (vedi **Figura 8b**): alla tensione V_{OL} (compresa tra 0 e 0.4V e tipicamente pari a 0,2V), presente sulla linea d'uscita TTL quando la Riga corrispondente è a livello basso, deve essere aggiunta la caduta di tensione ai capi del diodo, in forte conduzione; per garantire che la differenza di potenziale, somma delle due, sia compatibile con la tensione V_{IL} (compresa tra 0 e 0.8V, necessaria perché venga riconosciuta come livello basso dalla linea d'ingresso di POTRB) è consigliabile utilizzare diodi di tipo Schottky che (come quelli al Germanio, ormai quasi introvabili) hanno una tensione di soglia decisamente inferiore a quella di un comune diodo al silicio.
Probabilmente si tratta di un romantico eccesso di zelo ma sta di fatto che la caduta di tensione di un diodo al silicio sia compresa tra 0,6V e 1,7V mentre quella di un

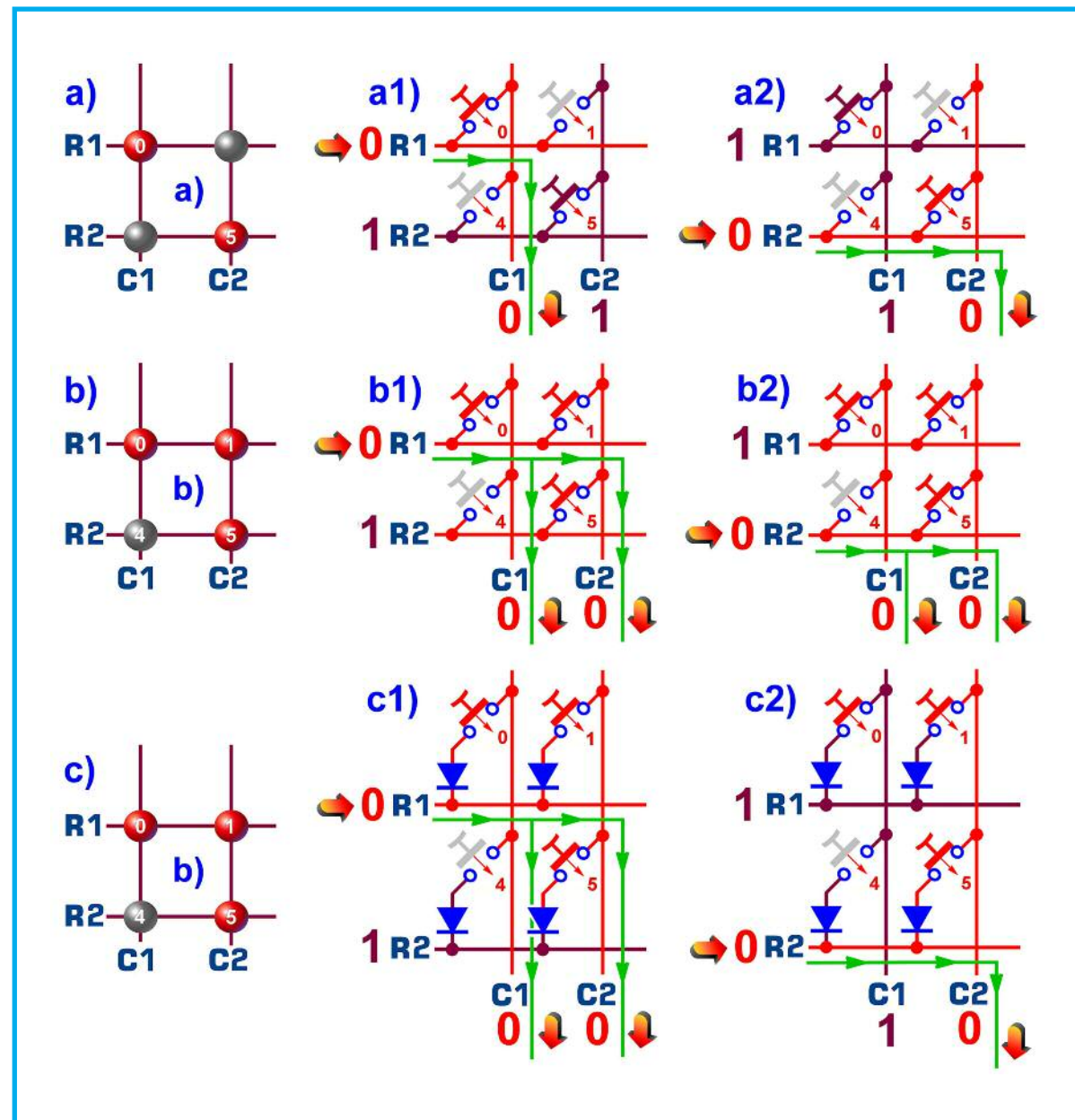



Figura 9 - Fenomeno di Key Gosting e soluzione anti-Gosting, nelle Tastiere a Matrice

diodo Schottky va da 0,25V a 0,45V, garantendo tra l'altro una velocità di commutazione superiore e una migliore l'efficienza complessiva.
In conclusione, la scelta di un tastierino è certamente meno impegnativa di quella di una tastiera di altro tipo, di solito caratterizzata da un numero di tasti importante e quasi sempre chiamata a garantire un servizio affidabile anche in presenza della pressione di più di un tasto; si pensi alla tastiera di un PC (nella quale è spesso necessario leggere anche tre tasti contemporaneamente, di solito Ctrl, Alt, Del o Shift con altri ..) o a quella di uno strumento musicale.
L'unico parametro utile ai nostri progetti (a parte quello estetico) è il ritardo dovuto al rimbalzo dei tasti (Key bouncing delay), funzione della qualità del contatto, computabile tra 2 e 50 ms e sempre indicato nel datasheet della Keypad.
Se bisognasse garantire il servizio di una tastiera di altro tipo è necessario conoscere altri parametri, in aggiunta al precedente: a) la velocità di campionamento (polling rate), cioè la frequenza con cui la logica programmabile in attesa può ricevere informazioni attendibili dalla tastiera: tenendo conto del fatto che il tempo umanamente possibile tra due pressioni non può essere inferiore a 200 0 300 ms, questo criterio di valutazione può ritenersi marginale e comunque inutile, oltre il suo valore tipico di 125 Hz; b) il tempo di risposta (response time), sostanzialmente lo stesso dato appena descritto, ottenuto invertendo in valore precedente: valori inferiori a (1/125 Hz)=8 ms sono dunque irrilevanti, per una tastiera; c) il numero KRO (Key Rollover), cioè il numero di tasti che, premuti contemporaneamente, possono essere riconosciuti dalla tastiera; la maggior parte delle Keypad sono 2-KRO o 3-KRO, ma una normale tastiera per PC deve essere almeno 6-KRO; esistono anche versioni NKRO (per esempio le tastiere musicali e quelle più costose) nelle quali ogni tasto è analizzato in modo indipendente ed è sempre rilevato correttamente dall'hardware, qualunque sia il numero di quelli che risultano premuti.
Nelle tastiere a Matrice di tipo economico (contenenti solo molti tasti, senza altro hardware) può manifestarsi il fenomeno di Key Ghosting, legato al numero espresso dal parametro Key Rollover; pur essendo a margine delle finalità di questo articolo, ritengo importante esserne a conoscenza.
Esso si palesa sempre quando più tasti sono premuti contemporaneamente e consiste nel fatto di ritenere premuto un pulsante (per questo detto "fantasma") che non lo è, a causa del percorso che la corrente può seguire attraverso i contatti effettivamente chiusi; questo evento è difficilmente associabile ad una Keypad, proprio perché su essa di solito si preme solo un tasto alla volta, consentendo al suo programma di gestione di localizzare senza ambiguità la combinazione Riga/Colonna che corrisponde all'unico contatto chiuso.
Anche se i tasti premuti sono due la rispettiva codifica non crea alcun problema: se essi sono su Righe e Colonne differenti (come in **Figura 9a**) la scansione rileverà, ancora senza ambiguità, prima un contatto e poi l'altro, annotando le rispettive coordinate, nell'esempio: R1_C1 e R2_C2; ma il risultato non cambia anche se i due tasti sono sulla stessa Riga o sulla stessa Colonna: solo i due contatti chiusi saranno rilevati in sequenza e codificati con certezza, per esempio con le coordinate R1_C1 e R1_C2 o con quelle R1_C1 e R2_C1.
Ma veniamo alla situazione critica: la **Figura 9b** mostra il caso in cui risultano premuti i pulsanti "0", "1" e "5" e, di certo, il programma di scansione ne rileverà la chiusura segnalando le rispettive coordinate valide, R1_C1, R1_C2 e R2_C2; il fatto spiacevole è che esso rileverà anche il tasto "4", decisamente non premuto, segnalando anche le coordinate R2_C1 di un contatto inesistente!
Quando 3 pulsanti legittimamente premuti sono collocati su tre angoli di una figura quadrata o rettangolare, anche i contatti del pulsante che occupa l'angolo rimanente risultano in corto, anche se il tasto stesso è ancora aperto.
La cosa è evidente osservando la **Figura 9b**, nella quale: b1) se la massa è fornita su R1 (con R2 ancora a 1) lo 0 sarà rilevato in sequenza prima su C1 e poi su C2, come ci si attende e come deve essere; b2) se lo 0 è fornito su R2 (con R1 a 1) es-

so sarà rilevato su C2, come ci si attende, ma anche su C1, sebbene il tasto "4" sia aperto: i suoi contatti sono comunque entrambi a 0 (in corto) per la situazione di continuità elettrica imposta anche su questa colonna dalla chiusura degli altri tasti. Se non si prendono provvedimenti il programma di gestione invia al processore un'informazione sbagliata; per questa ragione i costruttori delle tastiere professionali obbligano il suo controller a generare un segnale d'errore (Key Blocking) che impone alla CPU, almeno per le combinazioni di tasti "a rischio Key Ghosting", di non riconoscerne la pressione.

Per assurdo, questo tipo di prevenzione genera un ulteriore problema: se, con 3 tasti premuti, venisse operata anche la pressione del quarto (non rilevata, in virtù delle logiche di Key Blocking) non verrebbe rilevato nemmeno il suo rilascio, che continuando ad essere considerato premuto, verrebbe mascherato (Key Masking).

La soluzione più semplice per risolvere ogni problema di Ghosting e/o Blocking è quella di dotare ogni tasto di diodi in serie, così da impedire il passaggio di corrente tra tasti adiacenti.

La **Figura 9c** ripropone il caso in cui risultano premuti i pulsanti "0", "1" e "5", ma ora è presente un hardware anti-Ghosting: c1) con R1=0 e R2=1 non cambia nulla: lo 0 passa su entrambe le colonne è verrà rilevato in sequenza prima su C1 e poi su C2; c2) con R1=1 e R2=0 lo 0 sarà ora rilevato solo su C2 (come deve essere, in virtù della pressione di "5") ma non più su C1, per la presenza dei diodi in serie ai tasti della R1, polarizzati inversamente.

La presenza di un diodo su ogni tasto rende più elevato il costo della tastiera a matrice e, tra l'altro, può essere utilizzata solo se di tipo meccanico, come le quelle musicali e quelle di maggior pregio per computer; ma in questo modo è garantita la lettura (sia in pressione che in rilascio) di tutti i tasti, anche contemporaneamente, qualunque sia la loro posizione nella matrice, e per questo (come anticipato) sono catalogate NKRO (N-Key Rolleover).

Naturalmente questo evento è decisamente improbabile, ma il coinvolgimento di un gran numero di essi non è così assurdo: basti pensare alle esigenze di un musicista quando produce un accordo a due mani, sulla sua piano keyboard ..

Per finire, torniamo ad occuparci di Keypad: ci sono altre tecniche di gestione che possono stimolare qualche attenzione; se il microcontrollore è dotato di un convertitore A/D sufficientemente veloce e preciso è possibile gestirla (in modo piuttosto originale) con una sola linea.

L'idea, documentata dai datasheet, consiste nel collegare dei resistori di precisione di valore pesato in serie a ciascuna Riga e a ciascuna Colonna della Matrice; in questo modo è possibile costruire un partitore resistivo in grado di generare una tensione di valore proporzionale alla posizione del tasto premuto, da applicare sull'ingresso dell'ADC.

L'altra soluzione, meno fantasiosa e decisamente interessante è quella di utilizzare circuiti integrati progettati specificatamente per il servizio di tasterini a matrice, come il 16-Key Encoder 74C922 e il 20-Key Encoder 74C923; ma di questo ci occuperemo la prossima puntata.

Prossimamente

La pneumatica con il cubloc

Un esempio di applicazione alla pneumatica dei microcontrollori con l'impiego del PC per l'elaborazione e la memorizzazione dei dati.

Costruiamo un generatore di onda quadra

Realizziamo assieme un semplice generatore di segnale ad onda quadra, utile in tante occasioni. Esso è capace di produrre segnali a bassa ed alta frequenza.

Attivazione temporizzata di un carico

Una semplice ed utile realizzazione che alla pressione di un pulsante attiva un carico e lo disattiva dopo alcuni secondi.

Indagine sui lettori

Aiutaci a conoscerti meglio!

Con il tuo aiuto riusciremo a offrirti una rivista sempre più in linea con le tue aspettative.

Compila online il questionario all'indirizzo www.farelettronica.com/survey

Per ringraziarti **per il tuo tempo e la tua cortesia, ti invieremo** gratuitamente **un bellissimo** eBook **del valore di** 14,52 euro!

ElettroQuiz n. 335/336

IN PALIO: abbonamento al club di Fare Elettronica

FACILE

Al circuito di figura è applicato un segnale sinusoidale di ampiezza 2V e frequenza 1KHz. Quali delle seguenti affermazioni sono vere?

a) Vout è una sinusoide di ampiezza 2V e frequenza 1KHz;
b) Vout è un'onda quadra;
c) Vout è un segnale pressoché nullo;
d) Vout è una sinusoide di ampiezza 1V e frequenza 1KHz.

DIFFICILE

Dato il filtro di figura, si determini la frequenza di taglio e il valore della tensione di uscita quando Vin è un segnale in continua e quando Vin è un segnale sinusoidale di frequenza 1MHz.

IN PALIO: Maker card

Le risposte ai quiz "facile" e "difficile" vanno inviate esclusivamente compilando il modulo su www.farelettronica.com/eq specificando la parola chiave "**Bode**".

Le risposte e i vincitori (previa autorizzazione) sono pubblicati alla pagina www.farelettronica.com/eq a partire dal 15 del mese successivo alla pubblicazione sulla rivista.

A tutti i partecipanti verrà assegnato un buono sconto del 10% (validità 3 mesi dalla data di assegnazione) utilizzabile per un prossimo acquisto su **www.ieshop.it**

di MARCO SOLIMANO

TAGLIOLA PER FULMINI

Un’idea per catturare un fulmine con la vostra fotocamera e calcolarne la durata

Questa idea mi è nata nel lontano 1998, ho realizzato questo circuito con l’intento di fotografare un fulmine utilizzando la mia mitica reflex Olimpus OM1. Nel circuito era prevista un’ elettrocalamita per attivare il pulsante di scatto, feci qualche foto, in una mi avvicinai all’obbiettivo prefissato,e constatai che avevo immortalato un bagliore dopo aver consumato un rullino da 24 scatti ! A quei tempi le fotografie si facevano su di una pellicola, portate poi a sviluppare dal fotografo, stampate su carta, e se le foto erano brutte si scartavano!
Le limitazioni erano i costi da sostenere tra acquisto pellicola, sviluppo e stampa, e poi si doveva attendere qualche giorno per avere il risultato. Questo metodo implicava una conoscenza almeno di base dell’arte fotografica, cioè esposizione, tempi di scatto diaframma, obiettivi ecc.!
Oggi con le moderne macchine fotografiche anche persone poco esperte possono ottenere ottimi risultati, in pratica si punta e si scatta ! Molte delle impostazioni che prima erano manuali ora sono completamente automatiche.
L’obiettivo che mi ero prefissato, era di non usare il classico metodo della posa B,ovvero otturatore aperto per un tempo dipendente dalla durata della pressione del pulsante di scatto, attendendo l’arrivo del fulmine.
Naturalmente questo si poteva fare solo di notte altrimenti la foto se scattata di giorno sarebbe stata “bruciata”.Un atro metodo è quello di impostare un tempo di posa di circa due secondi, diaframma chiuso, sensibilità ISO bassa, fare svariati scatti, giocando molto sulla fortuna di acchiapparne uno decente !
Con questo mio circuito invece si può scattare singolarmente anche di giorno, impostando opportunamente i tempi di posa, diaframma, ISO, ecc.. sicuri di ottenere un risultato come mostrano le foto in questo articolo
Dopo anni finalmente sono riuscito a fotografarne uno;il circuito l’avevo preparato già fine primavera 2012, attendevo le burrasche con emozione, ma l’anno scorso non so perché ma di fulmini davanti casa neanche uno, i temporali decenti giravano al largo da casa mia anche se in estate si facevano sentire sonoramente, e ricordo quando circa un paio di anni fa ne cadde persino uno nel mio orto ! circa 50m. !!!
Che botto !... Mi saltò la tv e la centralina per i pannelli solari che avevo costruito nel 2001.
Finalmente il 2 dicembre 2012 mi trovavo in laboratorio, quando sentii il temporale avvicinarsi, corsi a casa e posi l’attrezzatura sul terrazzo; fortunatamente non pioveva in quanto esso era al largo tra Portofino e Sori (GE).Collegato il circuito,impostata la macchina fotografica e posiziona-

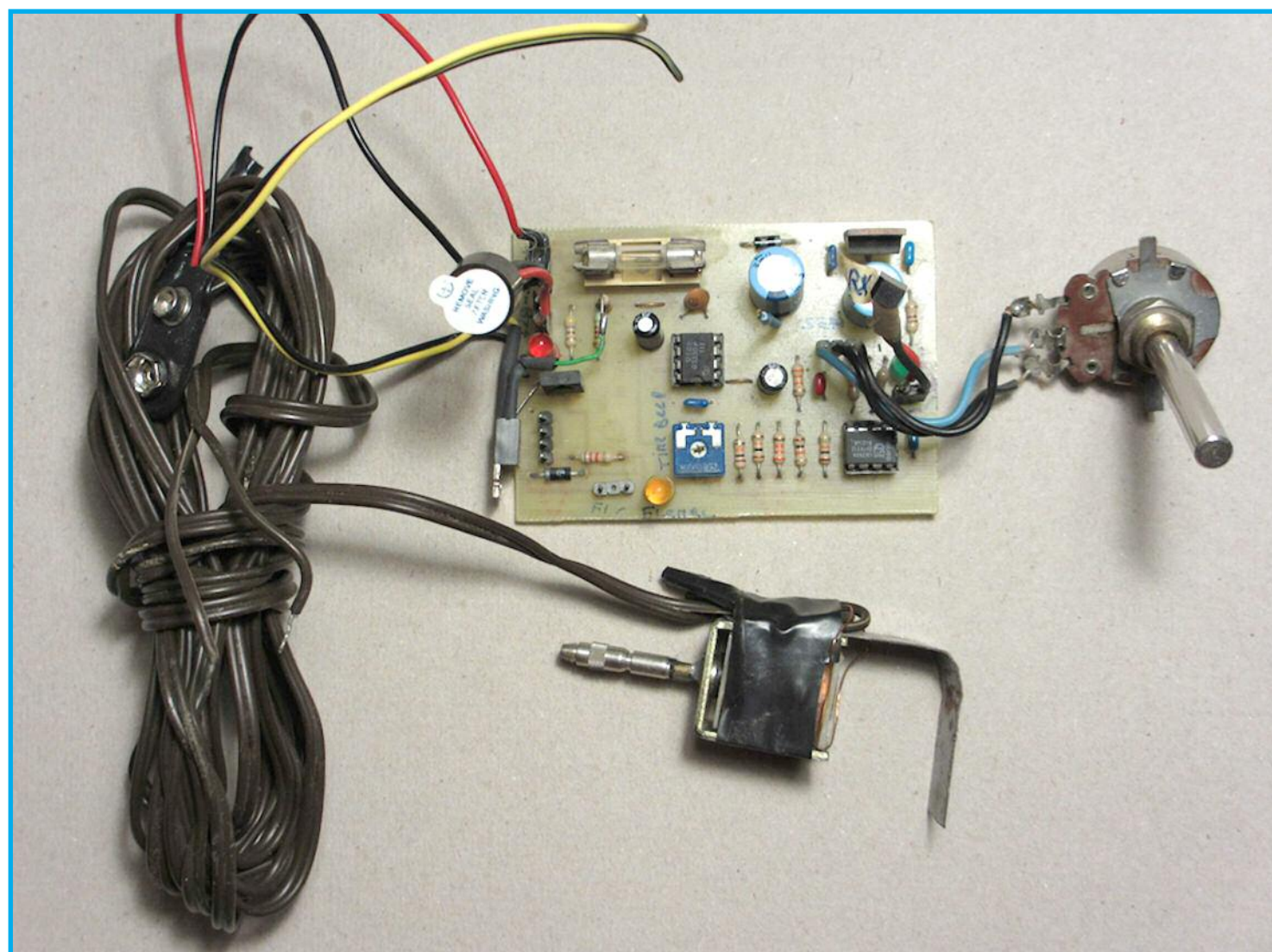



Figura 1: il primo prototipo della tagliola.

ta verso le nuvole il circuito la fece scattare immediatamente,erano circa le ore 17!Una giornata di vento, con cielo ad ovest perfettamente terso e a sud tutto nero,tipica giornata da "foto" con alti contrasti : appresi la sera su Facebook che una mia amica aveva fotografato un tornado al largo di Sori !
Il circuito costruito nel 98' prevedeva l'uso di una elettrocalamita che andava a premere il pulsante di scatto sincronizzato con il flash del fulmine, un sistema sicuramente lento rispetto alla velocità dello stesso, e poi occorreva un buon spunto di corrente !..usai questo circuito nel corso di questi anni per gioco con mia figlia per far suonare il cicalino quando ne cadeva uno !...preso !...da qui il nome di tagliola! Fig.1
Nella figura 1 e 2 si vede come era strutturato il primo circuito, invece nella figura 3 vediamo lo schema elettrico di quello attuale.

SCHEMA ELETTRICO

Dallo schema notiamo che l'alimentazione viene prelevata da una pila a 9volt, stabilizzata poi a 5 V per adeguare la tensione al microprocessore. Un alimentatore classico con U1 un IC stabilizzatore di tensione con 100 mA, più che sufficienti per questo scopo, ed i suoi condensatori di filtro e disaccoppiamento. Led 1 ci segnala che il circuito è ok.
U2 è un LM358 contenete 2 amplificatori operazionali, U2a lo usiamo per amplificare il segnale ricevuto dal sensore di luce infrarossa BPW41N, con R3 ne dosiamo la sensibilità cioè spostiamo il punto di soglia del sensore.
Infatti il diodo così collegato entra in conduzione al rilevamento di un flash infrarosso (vedi caratteristiche) cioè portando verso massa il pin invertente del primo operazionale, ottenendo in uscita pin1 circa la V di alimentazione.
Con il potenziometro R3 inviamo una soglia di minimo al pin 2, da zero volt a circa 900mV, è chiaro dunque che la potenza del flash influisce su questo parametro.
Grafico 1 minima sensibilità e Grafico 2 quasi massima sensibilità.
Il Grafico 3 mostra il segnale in ingresso del 1 operazionale (viola) e l'uscita del secondo operazionale (arancione). Si nota quando l'ingresso arriva a 0Volt l'uscita si porta a circa 5V. Il periodo di questa onda squadrata è la "durata" del fulmine, che viene poi calcolata dal microcontrollore e mostrato dal display. Questo è un esempio simulato con un accendino.
La sensibilità dipende anche dalla distanza del fulmine, dalla quantità di raggi infrarossi emessi, dall'orientamento del sensore, ad esempio io l'ho montato sul conte-

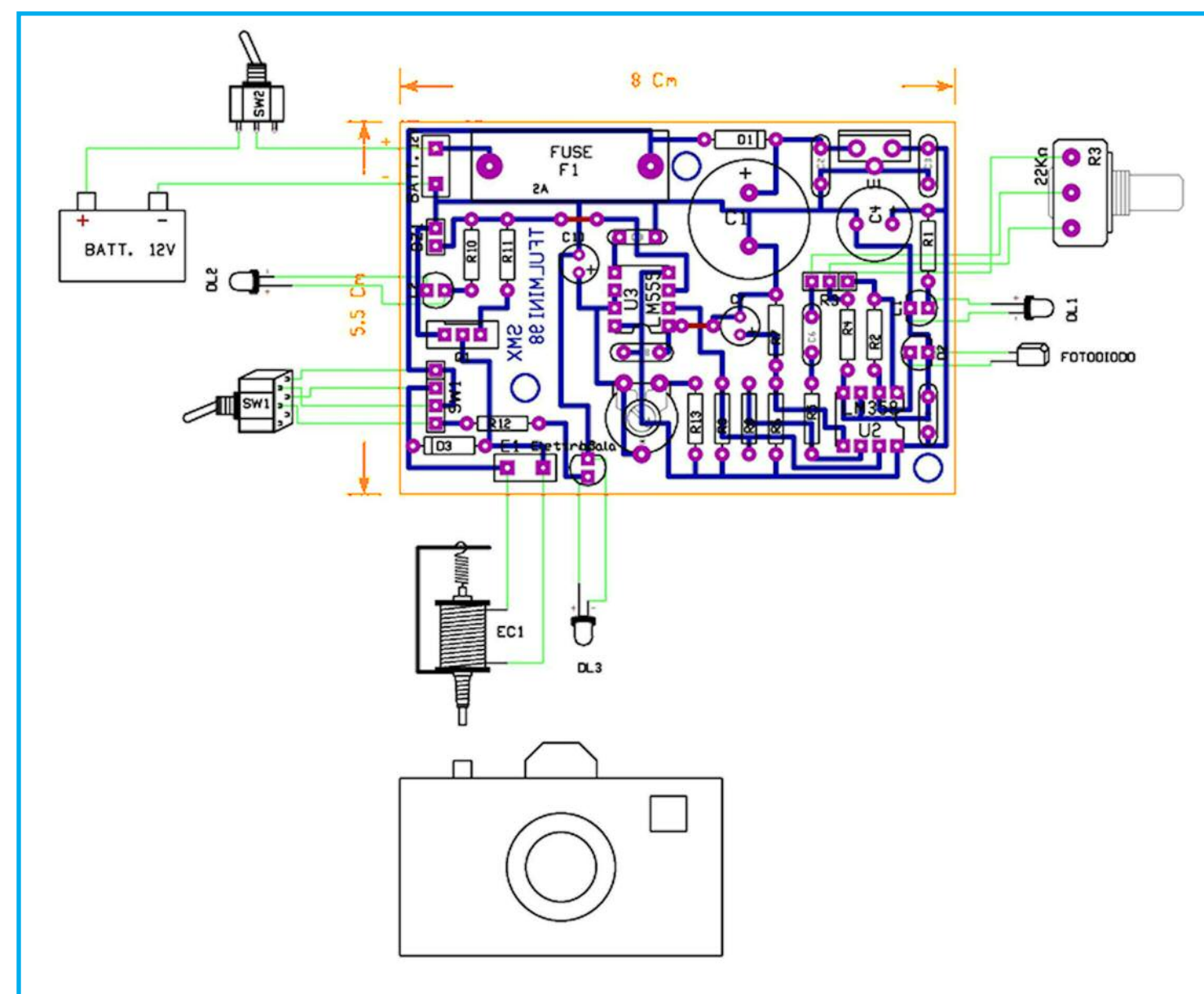



Figura 2: il piano di montaggio del primo prototipo.

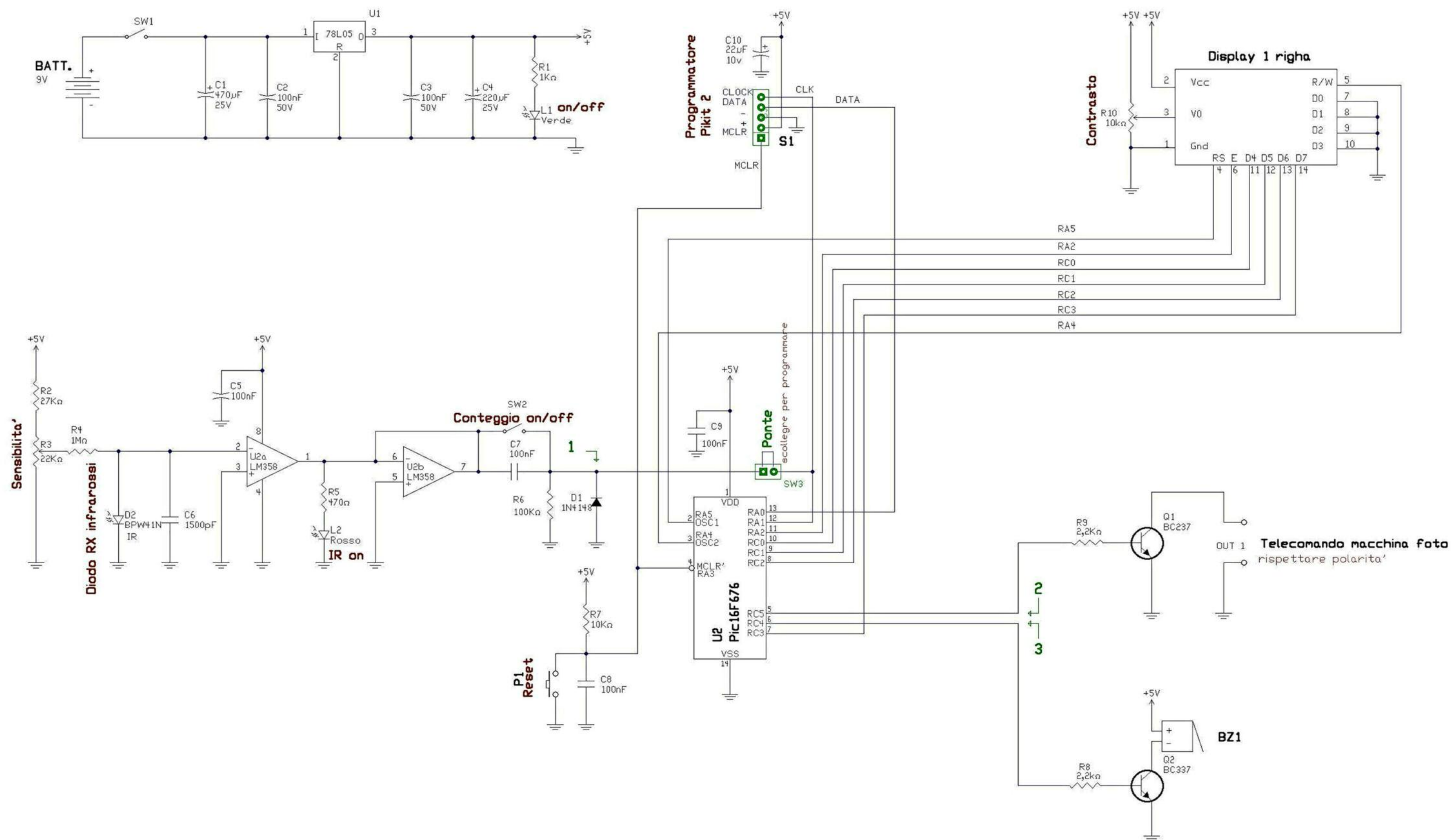



Figura 3: lo schema elettrico dell'ultima versione.

Elenco Componenti

R1 = 1 K	C1 = 470 uF 16V	U1 = 78L05
R2 = 27 K	C2 = 100 nF	U2 = LM358
R3 = 22 K Potenz. lineare	C3 = 100 nF	U3 = PIC16F676
R4 = 1 M	C4 = 220 uF 16V	Q1 = BC237
R5 = 470 Ohm	C5 = 100 nF	Q2 = BC337
R6 = 100 K	C6 = 1500 pF	Display 16 caratteri
R7 = 10 K	C7 = 100 nF	Led1 = verde
R8 = 2,2 K	C8 = 100 nF	Led2 = rosso
R9 = 2,2 K	C9 = 100 nF	Buzzer 5V
R10 = 10 K trimmer	C10 = 22 uF 16V	P1 = Pulsante
SW1= Interruttore slitta	SW2= Interruttore slitta	D1 = 1N4148
D2 = BPW41N	Batteria 9V	1 connettore strip 5 pin maschio
1 connettore strip 2 pin	Clip per batteria	
Contenitore ABS 130X60X29 con vano batteria		N.3 zoccoli per integrati

nitore inserendo una sorta di parabola, per limitare la luce entrante lateralmente, quindi rendendolo direzionale fig. 4.
Il led 2 ed il buzzer ci segnala il ricevimento del flash.
Il secondo operazionale, ci serve per squadrare il segnale in ingresso (grafico 3), il SW2 fa funzionare il rivelatore in due modalità, quando è aperto viene attivata solo la macchina fotografica e viene emesso un beep corto, mentre quando è chiuso si attiva subito la macchina fotografica e insieme misuriamo tramite un conteggio software la durata del fulmine, ovvero il tempo che rimane a livello alto l'uscita pin7 di U2, mostrandolo poi nel display.
Ruotando il potenziometro verso la massima sensibilità si accenderà il led ed verrà emesso un beep di breve durata (sempre se il sensore riceve un minimo di infrarossi anche da luce artificiale) se il SW2 è aperto, mentre se è chiuso si accende sempre il Led e verrà emesso un suono continuo avente come durata lo stesso periodo del livello alto del pin7
P1 azzera il display. Sono previste 2 uscite dal PIC, una comanda il Transistor Q1 ovvero simula un interruttore, mettendo in corto circuito il collettore con l'emettitore, azionando la macchina fotografica, Q2 per pilotare un buzzer.
In pratica il collettore lo collegheremo al polo centrale o punta di un jack da 3mm.
Le macchine fotografiche reflex, in genere hanno una presa jack per il telecomando manuale, ed è questo che useremo per fare le nostre fotografie. Il telecomando non è altro che un pulsante.
Io ho una Canon EOS 550D, ed è questa che ho usato per fare queste foto fig.5 e 6
Per fare queste foto è indispensabile l'uso di un cavalletto come fig.5
Dallo schema elettrico notiamo 2 connettori, uno a 5 poli usato per programmare il pic (collegato ad un programmatore per PIC es. Pickit2, ed il secondo 2 poli usato

Grafico 1: minima sensibilità

Grafico 2: quasi massima sensibilità.

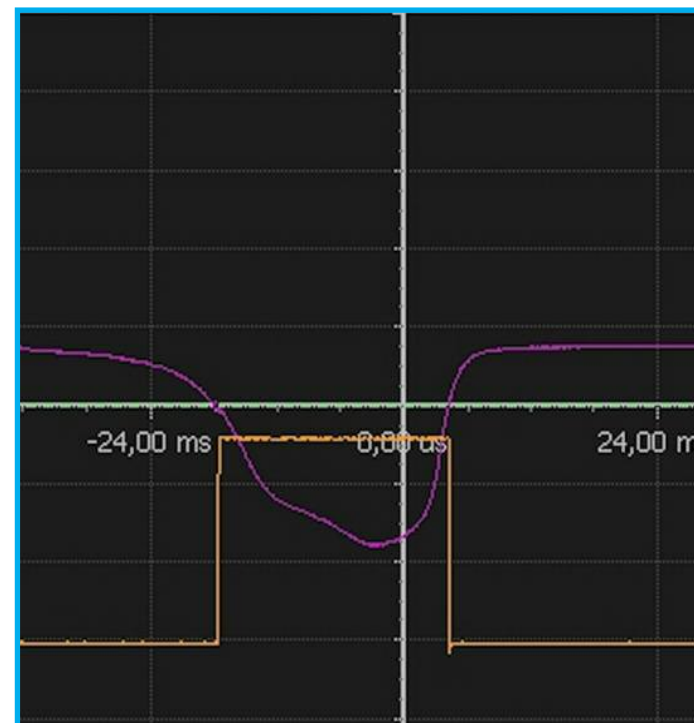



Grafico 3: il segnale in ingresso del 1 operazionale (viola) e l'uscita del secondo operazionale (arancione).

Figura 4: posizionamento del sensore

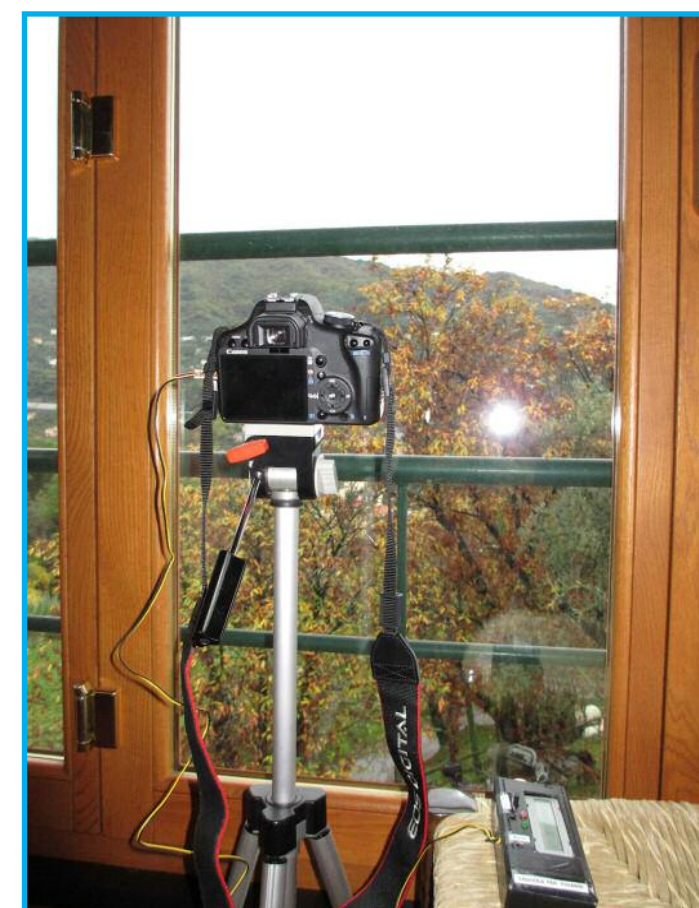



Figura 5: la macchina in posizione sul cavalletto

come ponticello, che va scollegato in fase di programmazione. Il trimmer R10 regola il contrasto del display, quello che ho utilizzato è una riga 16 caratteri, un rimasuglio del laboratorio. A chi non interessasse l'uso ed il montaggio del display con relativo software, può collegare direttamente i punti 1 con 2 e 3 come segnato nello schema elettrico. Le linee rosse che trovate nella fig.8 sono comuni ponticelli.

```
Tempo_fulmini_1.pbas
'---Configurazione porte -------------------
'EN, R/W, RS, D4, D3, D2, D1, sono configurati per pilotare il display
'Data, Clk, Mclr, sono configurazioni per la programmazione del pic
'RA.0=  DATA
'RA.1=  CLK   + IN1
'RA.2=  EN
'RA.3=  reset Mclr
'RA.4=  R/W
'RA.5=  RS

'RC.0=  D4
'RC.1=  D5
'RC.2=  D6
'RC.3=  D7
'RC.4=  BUZZER
'RC.5=  OUT1
```

Listato 1

```
'---Dichiarazione simboli e assegnazione porte -----
symbol in1    =porta.1
symbol Buzzer =portc.4
symbol out1   =portc.5

'---Dichiarazioni variabili---

'dim k as byte              'byte=8 bit = 255
'dim kk as word             'word=16 bit = 65535
dim conto as word
dim bit8 as byte
dim txt as string[5]       '5 variabili con nome txt1, txt2 ...

'***SETTAGGI  PRINCIPALI ******************
'---Principale---
main:                   'vedi data sheet convertitore analogico digitale
ANSEL=%00000000          'definisce ingressi analogici o digitali
                         'bit0, bit1, bit2, bit3, = AN0, AN1, AN2, AN3
                         'bit=0=digitale, bit=1=analogico
'adcon0=%0000111          'definisce canale convertitore analogico/digitale
                         'bit0=1=adon, bit1=1 abilitano convertitore
                         'bit2, bit3 = (00,AN0), (01,AN1), (10,AN2), (11,AN3) canali
cmcon=%00000111         'spengo comparatore an/dig
porta=0                 'azzera porta
trisa=%00001011         'defisce porta come ingresso e uscite (0=out, 1=in )
portc=0                 'azzera portc
trisc=%00000000         'definisce portc come ingresso e uscite (0=out, 1=in )
'****************************************
```

Listato 2

Figura 6: particolare della connessione al circuito

In figura 7 si vede il circuito stampato e le sue dimensioni, in figura 8 la disposizione componenti, in figura 9 il piano di montaggio, tutti gli schemi sono in formato Circad. Nella figura 10 vediamo il tutto assemblato, e nella fig.11 completo di contenitore, non proprio elegante ma visto che deve essere portatile ho dovuto comprimere il tutto, facendo anche un po' di errori di inscatolamento! La scatola e una comune 130x60x29 mm in ABS, con vano batteria 9V.

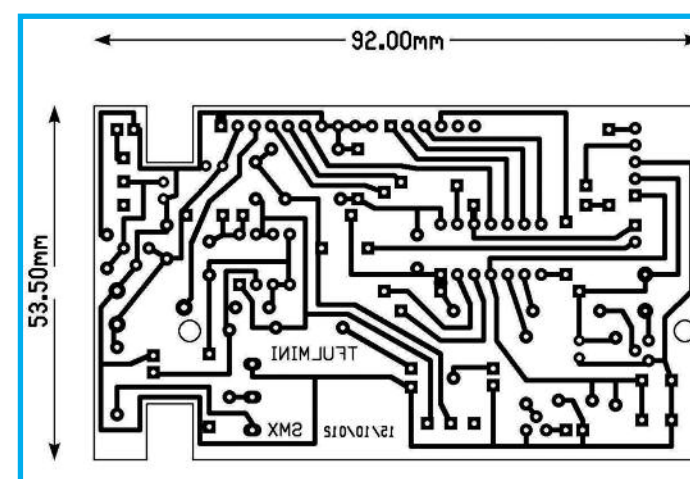



Figura 7: circuito stampato

SOFTWARE

Il microcontrollore è un PIC12F676 ha porte sufficienti per questo progetto ovvero, 7 porte per pilotare il display, 1 per il reset display, 2 per le uscite,1 per il sensore IR, 1 per la programmazione .(vedi datasheet)
Quindi 12 IN/Out, un oscillatore interno 4Mhz, tutti configurabili e 1 K di memoria questo è quello che ci interessa per questa applicazione, le altre caratteristiche non vengono utilizzate.
Come software di programmazione uso Mikrobasic V6, e programmo il Pic con Pickit2.
Con Questo Software la prima cosa da fare oltre a creare nuovo progetto, sono le impostazioni di base, quindi selezionare il tipo di Pic utilizzato, spuntare Mclre, Pwrte_off, Wdt_Off, Intrc_Osc_Noclockout. Vedi figura 12.
Dal listato dopo l'intestazione con nome e descrizione sommaria troviamo la configurazione delle porte, vedi listato 1.
Questi dati non sono compilati, li ho scritti solo per promemoria, la libreria di Mikrobasic 6 prevede 2 modalità di configurazione del display, 1) Lcd_init (portb), si può definire quale porta usare es. portc,ed i relativi dati, rs, r/w, en, sono per default già configurati.

- D7 port.7
- D6 port.6
- D5 port.5
- D4 port.4
- E port.3
- RS port.2
- RW port.0

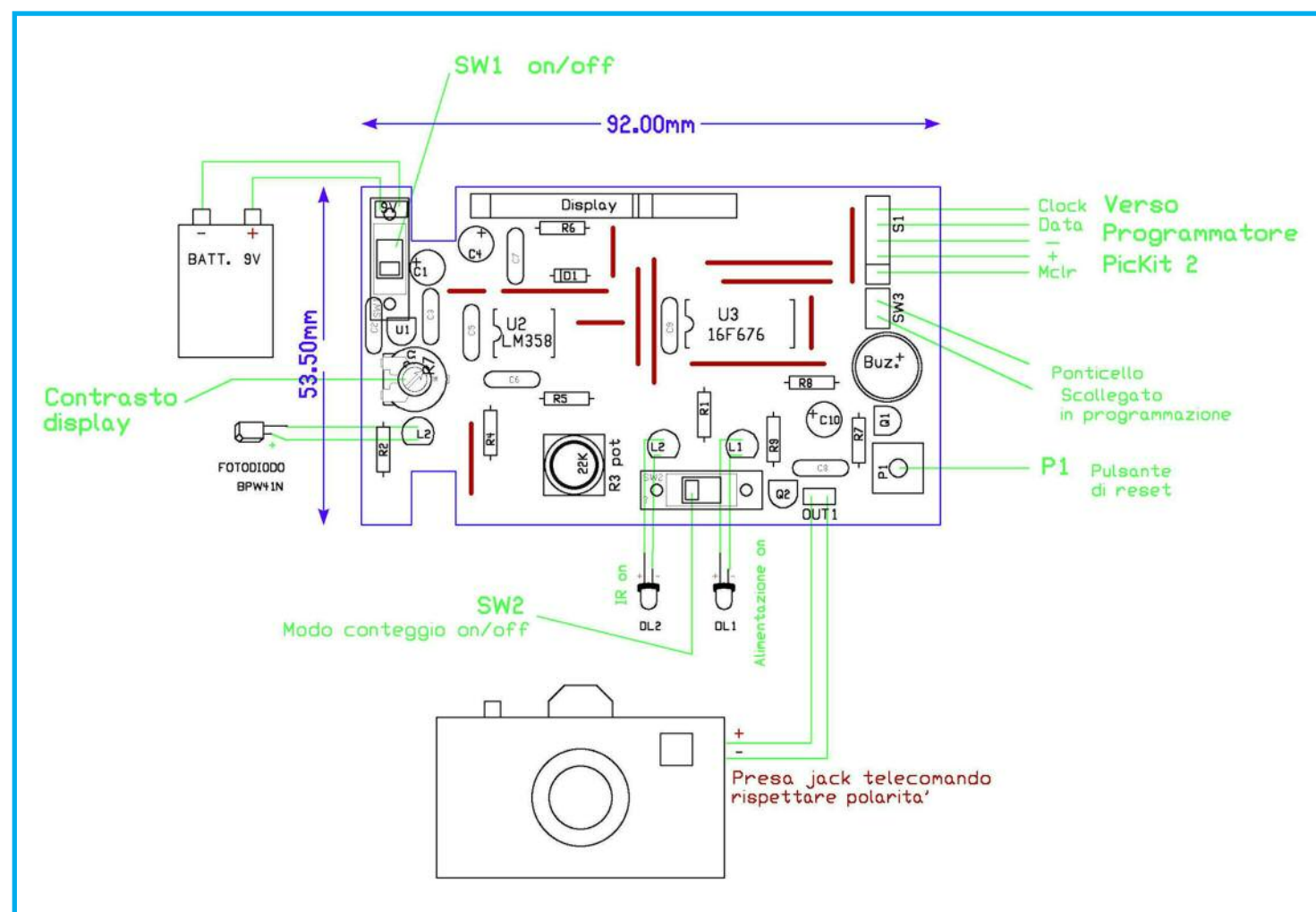
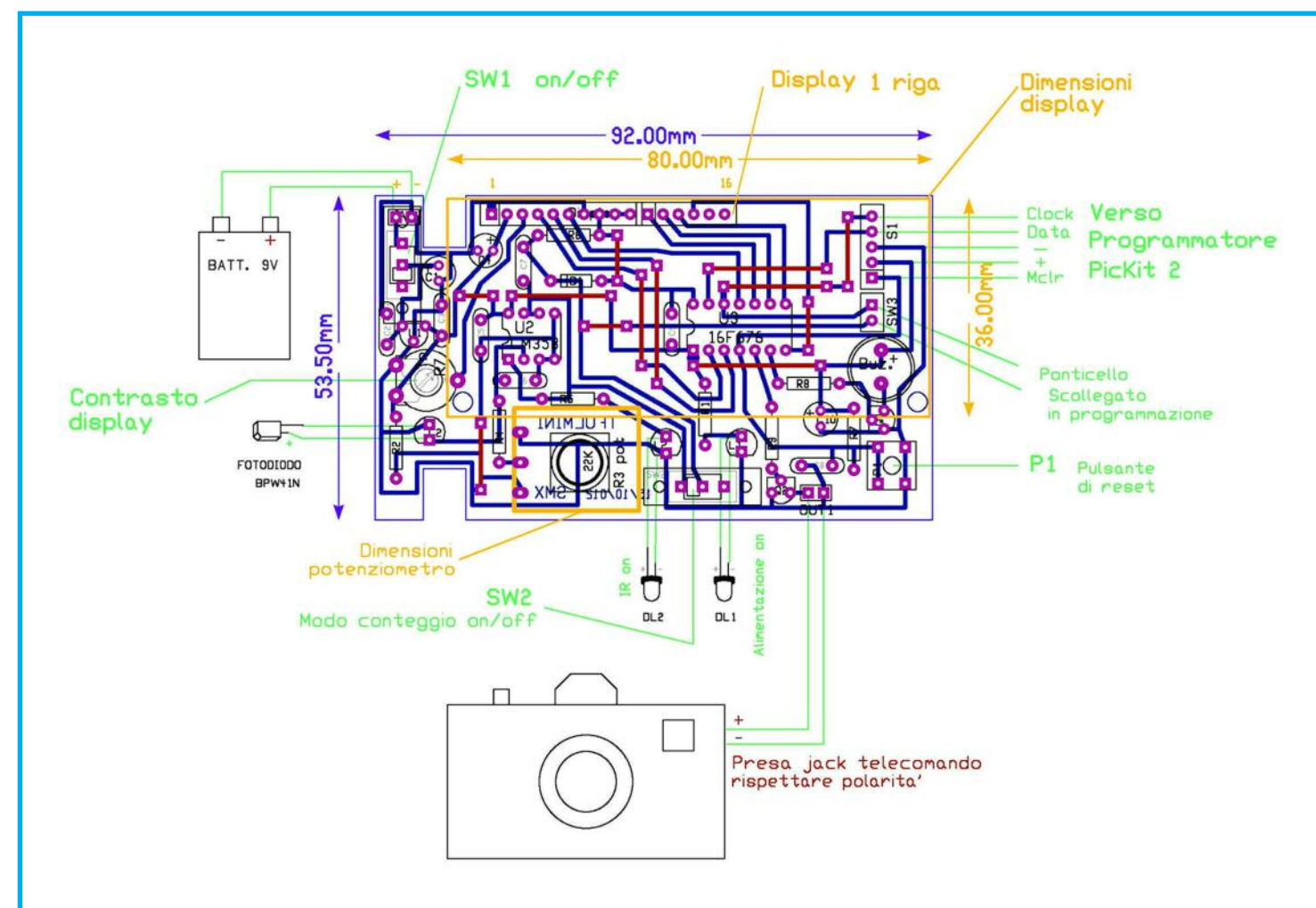



Figura 8: disposizione dei componenti

Figura 9: montaggio

2) Usando invece Lcd_Config, si può definire quale porta usare per i dati e quale per i controlli, e definire quali "pin" per la giusta configurazione.
es. Lcd_Config(PORTD,3,2,1,0,PORTB,2,3,4) si nota che le porte sono differenti, e differenti I pin attribuiti alle funzioni, bisogna solo rispettare la sequenza (quelli segnati in rosso nell'esempio).
Lcd_Config(**dim byref** data_port **as byte**,**dim** db3, db2, db1, db0 **as byte**,**dim byref** ctrl_port **as byte**,**dim** rs, ctrl_rw, enable **as byte**)
Nel mio software, ho utilizzato Lcd_config, perchè la port_a ha 6 porte, e la port_c idem, quindi non potevo usare un intero byte di controllo. Se volete potete utilizzare un altro PIC riconfigurando le porte.

Seguendo il listato 2, l'ingresso, l'uscita, e il pulsante di reset vengono associati alle rispettive porte e pin.
Si definiscono le variabili, che utilizziamo e poi i settaggi principali.
Dalla linea 50 Main configuriamo le porte in digitale, si spegne il comparatore analogico /digitale, azzero la port_a, definisco quali porte IN e Out, azzero port_c, e definisco di nuovo gli In/Out.
Seguendo dalla linea 64, metto un'attesa per stabilizzare l'alimentazione, poi configuro il display con le rispettive porte di comunicazione.
La linea 70 cancella il display. Dalla 72 inizia il ciclo del programma vero e proprio, quando arriva un flash, livello alto su porta .1, accendo subito l'uscita e il buzzer, e poi ne conteggio la durata (listato 3). Adesso dalla linea 84 Soft.4, trasferiamo questo conteggio al display, per fare questo bisogna:
cancellare il display,
trasformare la variabile che contiene il conteggio in una stringa di caratteri,
inviarli uno ad uno al display,
inserire l'unità di misura "S." (secondi)
inserire il mio nome (potete sostituirlo con il vostro).
Dopodiché spengo il buzzer, creo un ritardo in modo che venga letto come chiusura pulsante sulla macchina fotografica (se troppo veloce probabilmente non lo legge), spengo l'uscita.
Attendo 1 secondo e ritorno all'inizio attendendo un nuovo fulmine.
Perché trasformo la variabile di conteggio in una stringa ?, semplice se avessi scritto sul display 02763 S. sarebbe stato letto un numero espresso male, se invece scrivia-mo 0.2763 S. è più intuitivo.
Nota: La stringa è una sequenza di caratteri, dove ogni carattere è rappresentato da un byte.
Es. " Marco 22" viene rappresentato come da tabella ASCII con:

M = 77 (ASCI)	= 0100 1101 (binario)
a = 97	= 0110 0001
r = 114	= 0111 0010
c = 99	= 0110 0011
o = 111	= 0110 1111
spazio = 32	= 0010 0000
2 = 50	= 0011 0010
2 = 50	= 0011 0010

Quindi quando inviamo il nome "Marco 22", sequenzialmente invieremo i livelli logici rispettivi, alla porta selezionata.
Per poter mandare il tempo misurato in cicli, al display bisogna scomporre questo dato come sopra,
usando la libreria di conversione di Mikrobasic, creando tante variabili contenente i vari caratteri, e poi inviarli al display.

Vedi linea 85 del listato 4. Quindi scomponiamo una variabile tipo word in altrettante variabili tipo byte.

NOTE

Le fotografie che son riuscito a fare quella sera non sono molte, in quanto da casa mi son spostato in auto in altro luogo, inseguendo la burrasca, ma così ho perso

```
  · delay_ms(200)        'attesa per stabilizzare
 65
  · '*****Configurazione porte per il Display*****
  · Lcd_Config(PORTC,3,2,1,0,PORTA,5,4,2)    'configurazione porte pic
  ·                                           'data_port -- db3, db2, db1, db0
  ·                                           'ctrl_port-- rs, ctrl_rw, enable
 70 Lcd_Cmd(Lcd_Clear)                        'cancello lcd
  · '*****INIZIO*****
  · inizio:
  · If in1=0 then goto inizio else        'se IN1=0 torno a inizio (attendo fulmine)
  ·    out1=1  buzzer=1                    'Altrimenti se IN1=1 accendo uscita e buzzer
 75    conto=0                             'conteggio la durata del livello IN1=1
  · conteggio:
  ·   if in1=1then        'inizio ciclo conteggio tempo fulmine
  ·      conto=conto+1
  ·      delay_us(75)    'per avere un ciclo di 100us mettere 87us calcolato con mikrobasic
 80      goto conteggio 'calcolato con l'oscilloscopio 75us trascurare ultima cifra
  ·   end if
  ·
```

Listato 3

```
  ·       Lcd_Cmd(Lcd_Clear)                 'cancello display
 85       WordToStrWithZeros(conto, txt)     'trasformo la variabile Word in stringa di 5
  ·                                          'caselle(1byte per casella)con zeri iniziali
  ·                                          'ogni byte, corrisponte ad un carattere nella
  ·                                          'tabella ascii ..vedi !
  ·       lcd_chr(1,1,txt[0])                'visualizzo il primo byte della stringa
 90       lcd_out(1,2,".")                   'inserisco il punto
  ·       lcd_chr(1,3,txt[1])                'visualizzo il byte1
  ·       lcd_chr(1,4,txt[2])                '..
  ·       lcd_chr(1,5,txt[3])                '...
  ·       lcd_chr(1,6,txt[4])                '...
 95                                          '1°riga lunga 1byte =8 caratteri
  ·                                          '2°riga lunga 1byte =8 caratteri
  ·       lcd_out(2,1,"S.")                  'per visualizzare alla colonna 9 del display 1 riga
  ·       lcd_out(2,6,"SMX")                 'per visualizzare alla colonna 9 del display 1 riga
  ·                                          'il mio nome
100       buzzer=0
  ·       delay_ms(100)                      'tempo fisso per aggancio macchina fotografica
  ·       out1=0                             'eventualmente entrata in standby
  · end if
  · fine:
105 delay_ms(1000)                           'attesa nuovo ciclo
  · goto inizio
  · end.
```

Listato 4

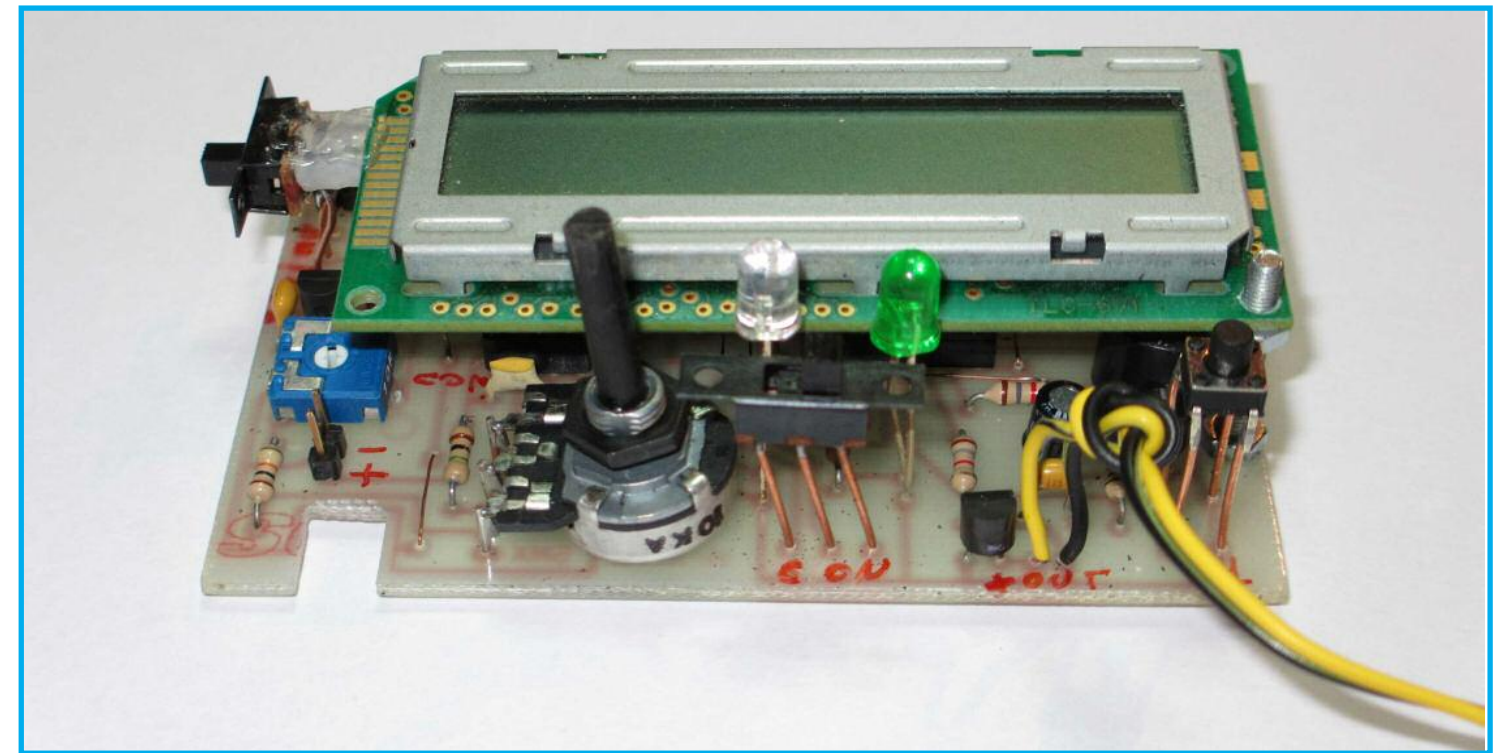



Figura 10: il prototipo assemblato

tempo ed il momento migliore!
Se di giorno risulta troppo sensibile, potete modificare questo parametro variando i valori del partitore R2 e R3.
Questo circuito è ancora in fase di test, in quanto collegato alla frequenza delle burrasche, lascio aperta la sperimentazioni a voi lettori.

La macchina fotografica deve essere impostata in modo manuale, bisogna disabilitare tutti gli standby, l'autofocus, lo stabilizzatore ottico (per chi ce l'ha), e tutti gli automatismi che rallenterebbero acquisizione.
Impostare il fuoco o all'infinito o sulla porzione di cielo dove si pensa che avvenga il flash, durante il giorno impostare l'esposi-

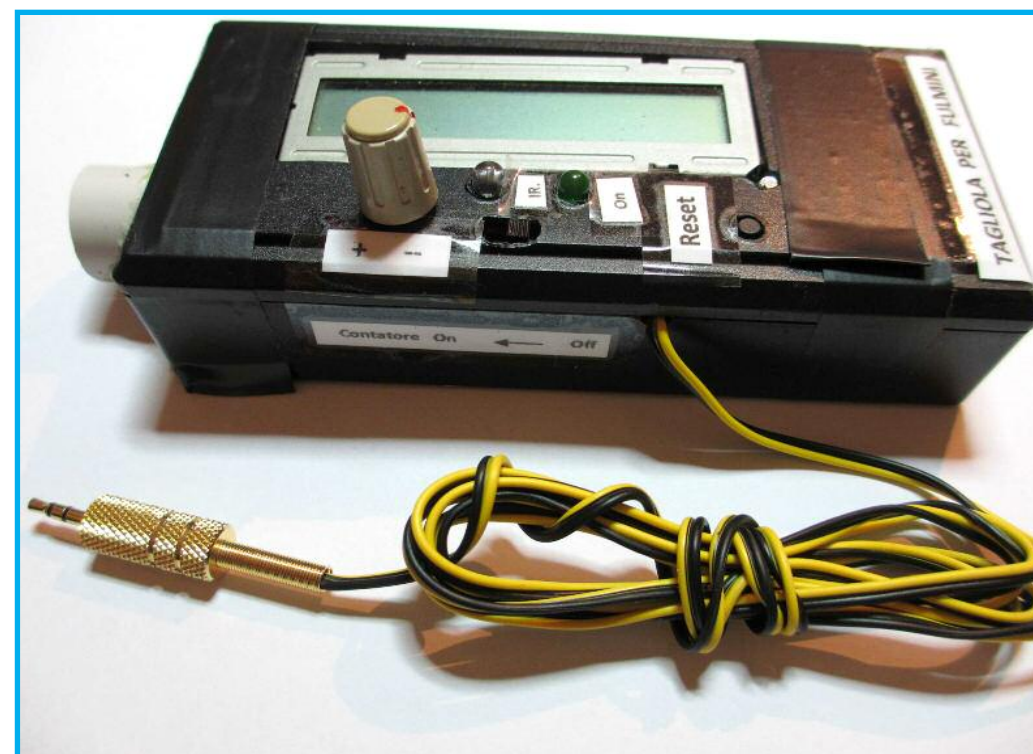



Figura 11: il dispositivo inserito nel contenitore.

zione adeguata, fate qualche prova.
La precisione della lettura tempo è da considerare “circa”, millisecondo + o - !
Nota importante, sapere quanto tempo intercorre dalla chiusura del switch sul telecomando al momento di acquisizione immagine, esempio se il fulmine dura più 35ms e la latenza della macchina fotografica è 30ms, lo catturerete certamente, altrimenti “buio”.
Fate un po’ di prove e magari per i meno esperti di fotografia guardate in rete dove potete trovare degli ottimi corsi ed esperienze riguardo le impostazioni luce, obiettivi, tempi di esposizione, automatismi ecc.
PS. State lontano dalle burrasche …!!!! Non esponetevi all’aperto troppo vicino, con cavalletti,telefonini, medaglioni, catenine ed ombrelli (sarebbero ottime antenne di ricezione) ecc… state chiusi in macchina o in casa, o notevole distanza! Meglio non aver fotografato un fulmine, che averlo **preso**!!!
Fatemi vedere le vostre catture, sono su Facebook, postatemi!

Figura 12: Schermata di impostazioni per la programmazione del PIC.

di VINCENZO SORCE

COMUNICAZIONE WIRELESS

CIRCUITO WI-COM-24

(parte prima)

La soluzione ottimale, a basso costo, per realizzare la comunicazione wireless fra una unità di comando trasmittente, sia essa PC, tablet o smartphone, purchè dotata di uscita USB, ed una unità ricevente di attuazione che pilota 24 utenze.

Nel panorama commerciale dell’elettronica esistono circuiti, piuttosto costosi, che realizzano comunicazioni wireless tramite la porta USB. Hanno del firmware residente e del software dedicato. Oltre ad essere costosi hanno un grande limite: comandano poche unità di uscita. Il circuito proposto invece, oltre a costare poco, consente di pilotare 24 utenze. E può andare oltre senza limite teorico. Infatti il limite è dato solamente dal numero di uscite del PIC adoperato.

Il circuito ricetrasmittente che abbiamo scelto consente la comunicazione fino a trecento metri all’aperto. Esso è la parte centrale di questo progetto ed ha la sigla SmartALPHA RF Transceiver (Figura 1).

Figura1: SmartALPHA RF Transceiver

Di esso parleremo diffusamente più avanti. Adesso analizziamo lo schema a blocchi del nostro circuito (Figura 2)

Dalla Figura2 è rilevabile un circuito master (letteralmente padrone) collegato alla porta USB del PC che, tramite una unità ricetrasmittente RX/TX, trasmette via radio i dati elaborati, tramite software, dal PC all’unità RX/TX della scheda SLAVE (letteralmente schiavo) che oltre ad attuare i comandi ricevuti, trasmette al master il controllo dell’attuazione.

Analizziamo più fondo il circuito master di Figura3. Esso è costituito:

- da un connettore USB che va collegato alla uscita USB del PC
- dall’integrato FT232RL
- dal TX/RX SmartALPHA Transceiver

L’integrato FT232RL è molto noto. Esso realizza l’interfaccia tra una porta USB ed un circuito che necessita di un collegamento seriale RS232-UART (Universal,

> ELENCO COMPONENTI
> CIRCUITO MASTER:
>
> - CNT1 connettore maschio tipo A USB per pcb
> - R1 470 Ohm – 1/4W
> - C1 0,1uF ceramico
> - LD1 Led D=3,5mm
> - U1 integrato FT232RL
> - U2 microcircuito SmartALPHA RF Trannsceiver

Figura 2: Schema a blocchi del circuito

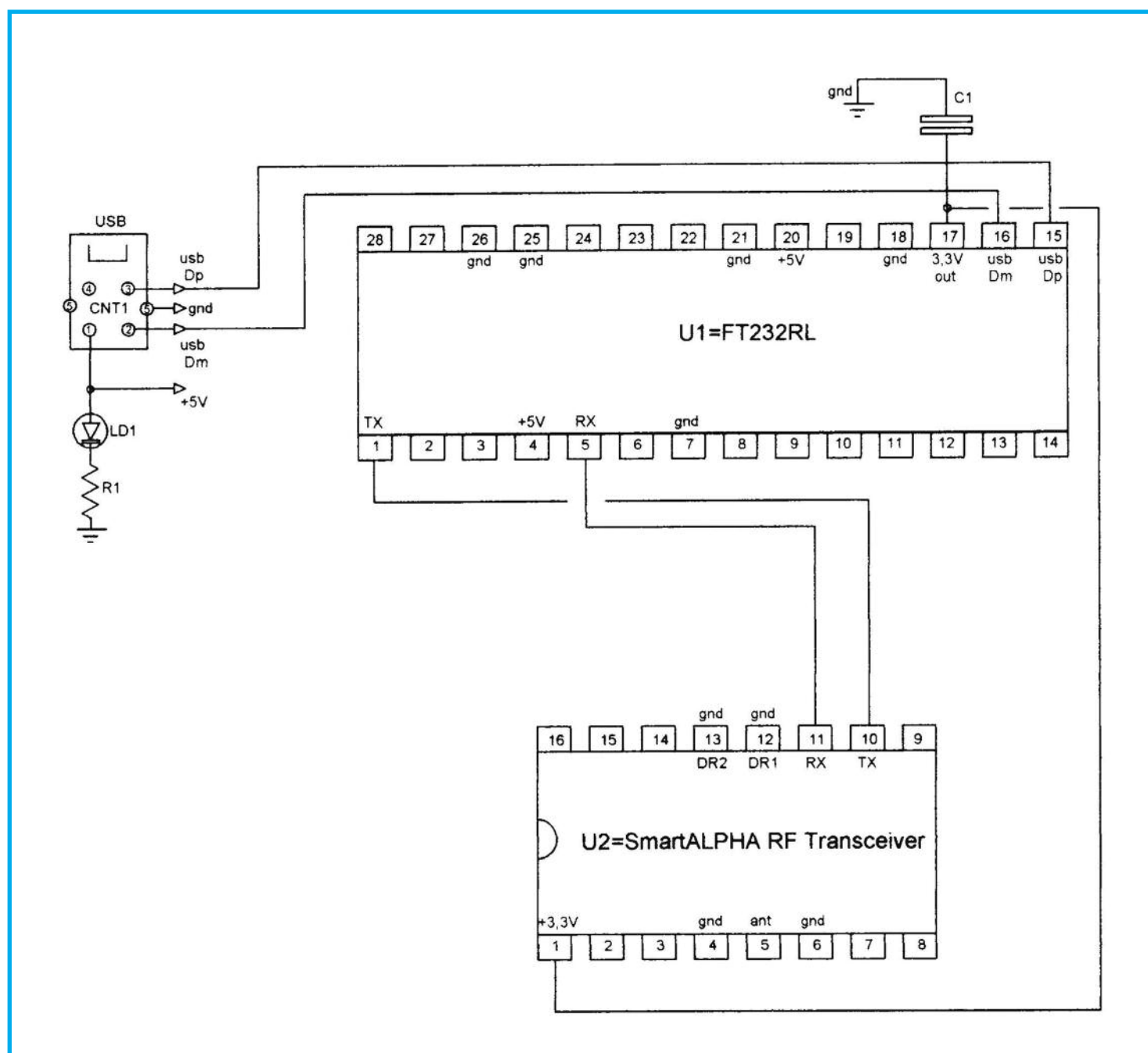
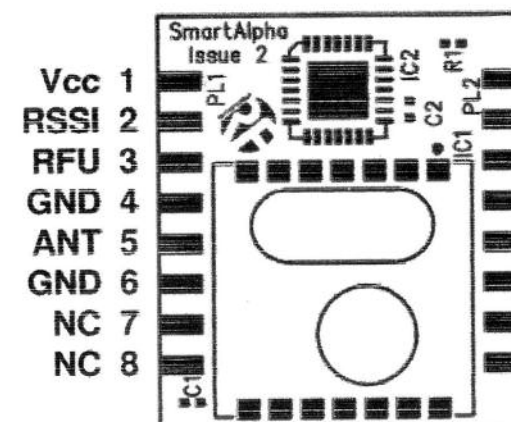



Figura3: circuito master

Asyncronus, Receiver/Trasmitter). E' un integrato del tipo smd (disposivo a montaggio superficiale).
Per chi non è esperto può essere un problema saldare su una scheda questo tipo componente. Ma vi assicuriamo che non è per niente difficile. Basta seguire queste semplici istruzioni:

- ricoprire leggermente di stagno le piste del PCB (circuito stampato) interessate
- posizionare il componente da saldare in modo da far combaciare i piedini dello stesso alle piste del pcb
- con una pinzetta (anche quelle adoperate per fissare la biancheria) bloccare il componente nella scheda (attenzione! Se non vi è corrispondenza ed è necessario spostare il componente, conviene prima allentare la presa della pinzetta per non provocare la distorsione dei piedini)
- infine passare uno strato continuo di ATTAK nei piedini ed attendere che questo si asciughi (fino ad allora non togliere la pinzetta)

SMARTALPHA RF TRANSCEIVER

Host Data Rate Selection

DR2 (pin 13)	DR1 (pin12)	Baud Rate
0	0	4,800
0	1	9,600
1	0	19,200
1	1	38,400

Pin Descriptions

Pin Number	Name	Type	Description
1	Vcc	Power	Positive supply voltage connection. Decouple with 100n ceramic capacitor to ground.
2	RSSI	Out	Pin is high when RSS is higher than the programmed threshold
3	RFU	-	Reserved for future use
4, 6	GND	Power	Connect to 0 volts.
5	Antenna	In / Out	Nominal 50 ohm input/output impedance capacitively isolated from the internal circuit.
7, 8, 14	reserved	-	Don't connect
9	PD	In	Power Down pin. Take low for low power standby mode.
10	Tx	In	Transmit data input from host controller. Data input to the transmitter can be directly interfaced to CMOS logic drive operating on the same supply voltage as the transceiver.
11	Rx	Out	Received data output to host controller (CMOS logic out) representing true data as supplied to the transmitter.
12, 13	Data Rate	In	Host Data Rate selection.
15	RTS	In	Logic '0' is Request To Send. Take low when the host is ready to send data to the module or is ready to receive data from the module.
16	CTS	Out	Logic '0' is Clear To Send. Taken high when the module is busy.

Electrical Characteristics

	Min.	Typ.	Max.	Units	Notes
DC Levels					
Supply voltage	1.9	3.3	3.6	V	1
Supply current (Transmit mode)		24	26	mA	
Supply current (Receive mode)		13	15	mA	
Supply current (Standby mode)		300		uA	
Data input/output high	Vcc-0.3		Vcc+0.3V	V	
Data input/output low	0		0.3	V	
RF					
Working frequency: 433MHz Module	430.24		439.7	MHz	2
868MHz Module	860.48		879.5	MHz	2
915MHZ Module	900.72		929.27	MHz	2
Receiver sensitivity		-105	-100	dBm	
Transmitter RF power out		4		dBm	
Frequency deviation		+/- 15		kHz	
GFSK manchester encoded data rate		86.2		kbps	
Operating temperature	-20		+80	Deg C	
Storage temperature	-40		+100	Deg C	
Dynamic Timing					
Power up to stable receiver data out		30		mS	
Power up to full RF out		30		mS	
Standby to Receive mode		1		mS	
Standby to Transmit mode		1		mS	

Notes

1. Supply voltage should have <10mV ripple.
2. The application operating frequency must be chosen to comply with the Short Range device regulation in the area of operation.

Figura4: TX/RX

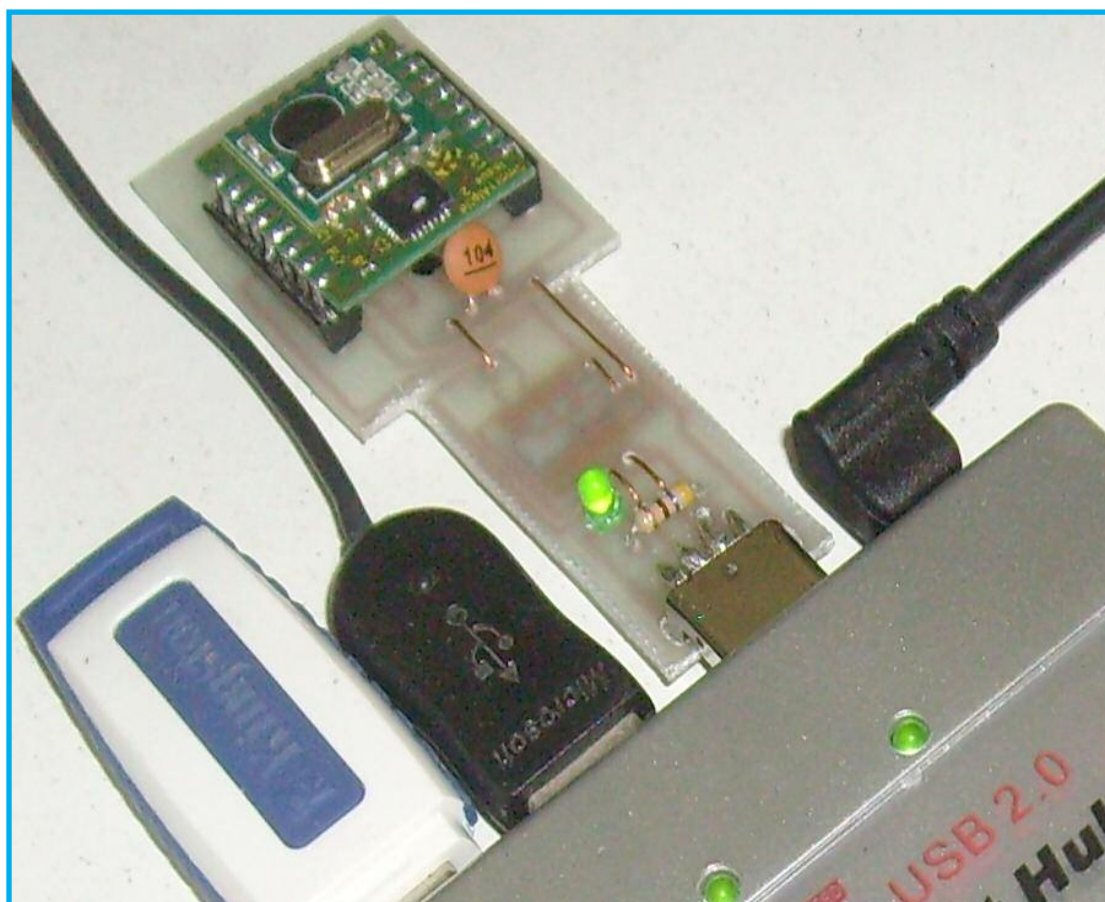



Figura5: foto del master realizzato

L'alimentazione va da +1,8V a 5,25V, perciò compatibile con quella fornita dalla porta USB che è di 5V.
Adesso è giunto il momento di occuparci della piccola scheda ricetrasmittente. A tale scopo analizziamo quanto rilevabile dal relativo datasheet riportato in Figura4:
La pedinatura, consistente in due file di 8 pin (piedini), a passo di integrato dip (Dual In-line Package, cioè doppia fila di piedini allocati nei fori della scheda), perciò di facile installazione. Necessita di una alimentazione che va da 1,9V a 3,6V. Poiché la tensione di alimentazione fornita dalla porta USB è di 5V, occorre utilizzare la tensione di 3,3V che fornisce l'integrato FT232RL al pin 17. I pin che noi collegheremo sono i seguenti:

- 4, 6 GND (ground, terra)
- 10 TX – trasmissione dati
- 11 RX – ricezione dati
- 5 – antenna – nel nostro caso non installata
- 12 e 13 GND

A proposito dei pin 12 e 13, osservando la tabella "Host Date Rate Selection" della Figura4 si evince che noi abbiamo scelto di trasmettere a 4.800 bps (bit per secondo). Il circuito è stato provato anche collegando i pin 12 e 13 a +3,3V (livello logico 1) e settando, quindi, la frequenza di trasmissione a 38400 bps, ed ha funzionato perfettamente.
A proposito dell'antenna vi è da dire che nel caso in cui la trasmissione avviene per esempio all'interno di in un appartamento, allora non occorre. Viceversa la si può facilmente costruire con un filo, oppure acquistarla. Nel primo caso il datasheet del componente dà le istruzioni occorrenti per realizzarla.
In Figura5 la foto del master realizzato

ELENCO COMPONENTI
SCHEDA SLAVE:

- da R1 a R24, R27: 1 kOhm 1/4W
- R25: 4,7 kOhm 1/4W
- R26: 680 Ohm 1/4 W
- R28: 1,2 kOhm 1/4 W
- C1: 0,1 uF poliestere
- C2: 1000uF – 25V
- C3, C4: 22pF ceramico
- C5: 100uF – 25V
- D1: diodo 1N4007
- Da LG1 a LG8: led D=3,5mm giallo
- Da LV1 a LV8: led D=3,5mm verde
- Da LR1 a LR8: led D=3,5mm rosso
- XT1: quarzo 20MHz
- P1: pulsante n.a. (normalmente aperto)
- CNT1: connettore di alimentazione per pcb
- M1, M2, M3: connettori femmina per pcb a passo dip
- U1: integrato regolatore di tensione uA7805
- U2: integrato regolatore di tensione LM317
- U3: microcircuito SmartALPHA RF Trannsceiver
- U4: integrato 40 pin PIC16F877A

Adesso occupiamoci della scheda "slave":
Mentre per il circuito "master" abbiamo utilizzato la tensione di +5V fornita dalla presa USB e della tensione di +3,3V fornita dall'integrato FT232RL, in questo caso abbiamo dovuto provvedere ad un circuito di alimentazione. Esso è stato così realizzato:

- alimentatore esterno che fornisce la tensione raddrizzata o continua da 12V
- integrato uA7805: regolatore di tensione che ha in ingresso +12V ed in uscita +5V
- integrato LM317 : regolatore di tensione con ingresso +12V ed uscita +3,3V

L'integrato LM317, a differenza dell'integrato uA7805, non dà un uscita di tensione determinata, la sua grandezza è regolata dal valore delle resistenze collegate ai suoi pin.
Analizziamo lo schema di figura6:

- Il segnale Radio trasmesso dal circuito master che esegue il comando ricevuto dal software installlato nel PC, con una portante nel nostro caso di 433MHz, ma, come si evince dal datasheet, può essere di 868MHz o 915MHz, a seconda della versione utilizzata, viene ricevuta dal circuito ricetrasmittente U3
- Dopo averlo processato e rilevato (cioè dopo aver estratto la parte digitale a 4.800 bps), tramite Tx, pin 11, viene trasferito su RA4, pin 6 dell'integrato PIC16F877A
- Grazie al firmware residente nel PIC si avrà il condizionamento delle uscite dello stesso, in sintonia con il segnale trasmesso
- Ogni volta che le uscite commutano, comunicano, tramite RA5, pin7 all'integrato U3, tramite Tx, pin 10, l'avvenuta commutazione
- Il suddetto integrato trasmette via radio il segnale che viene ricevuto dal circuito master
- Quest'ultimo, collegato al PC, grazie al software installato, mostra nella videata del programma, relativamente al pulsante interessato, lo stato di ON o OFF

Figura 6: scheda “slave”

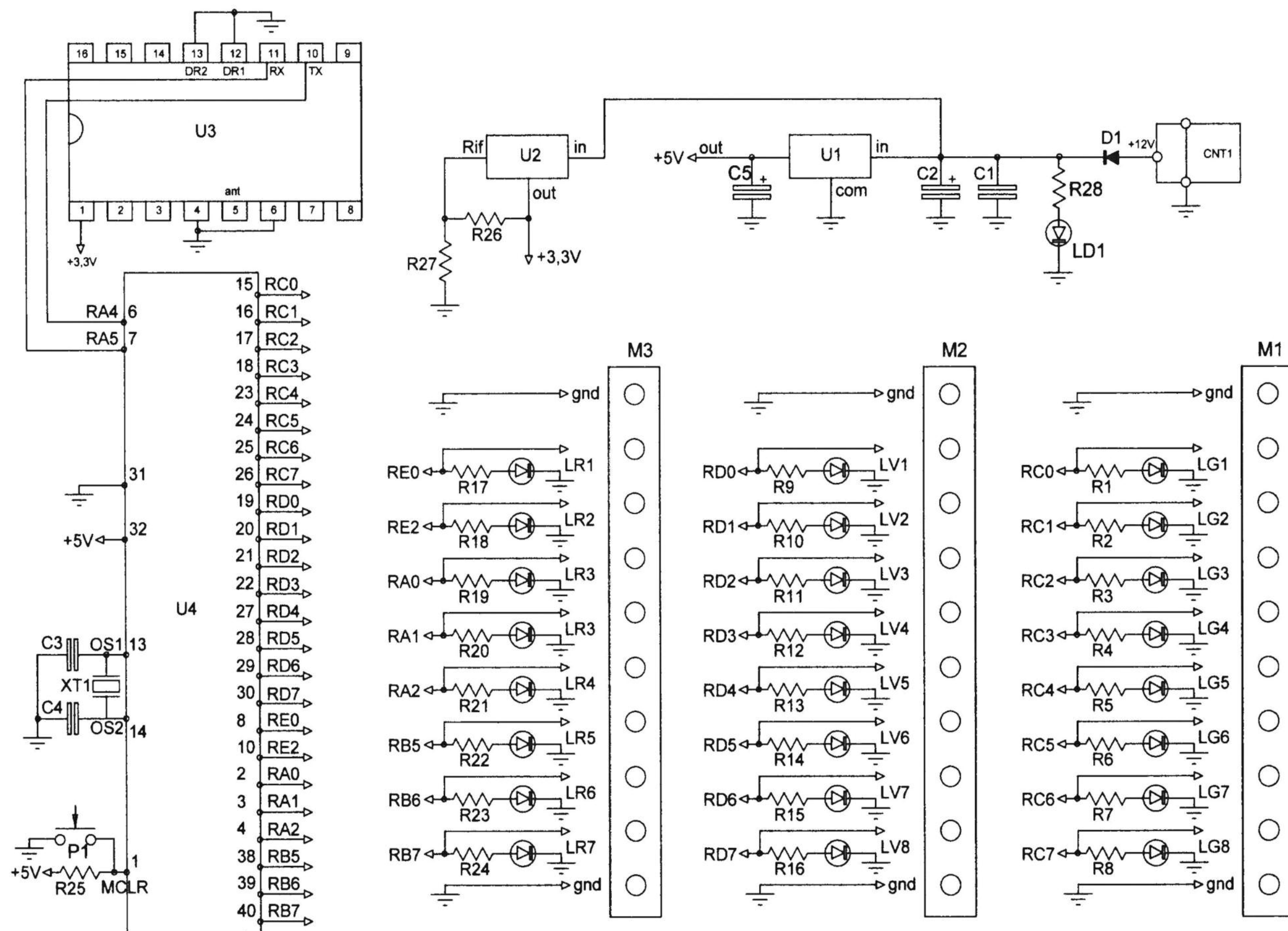
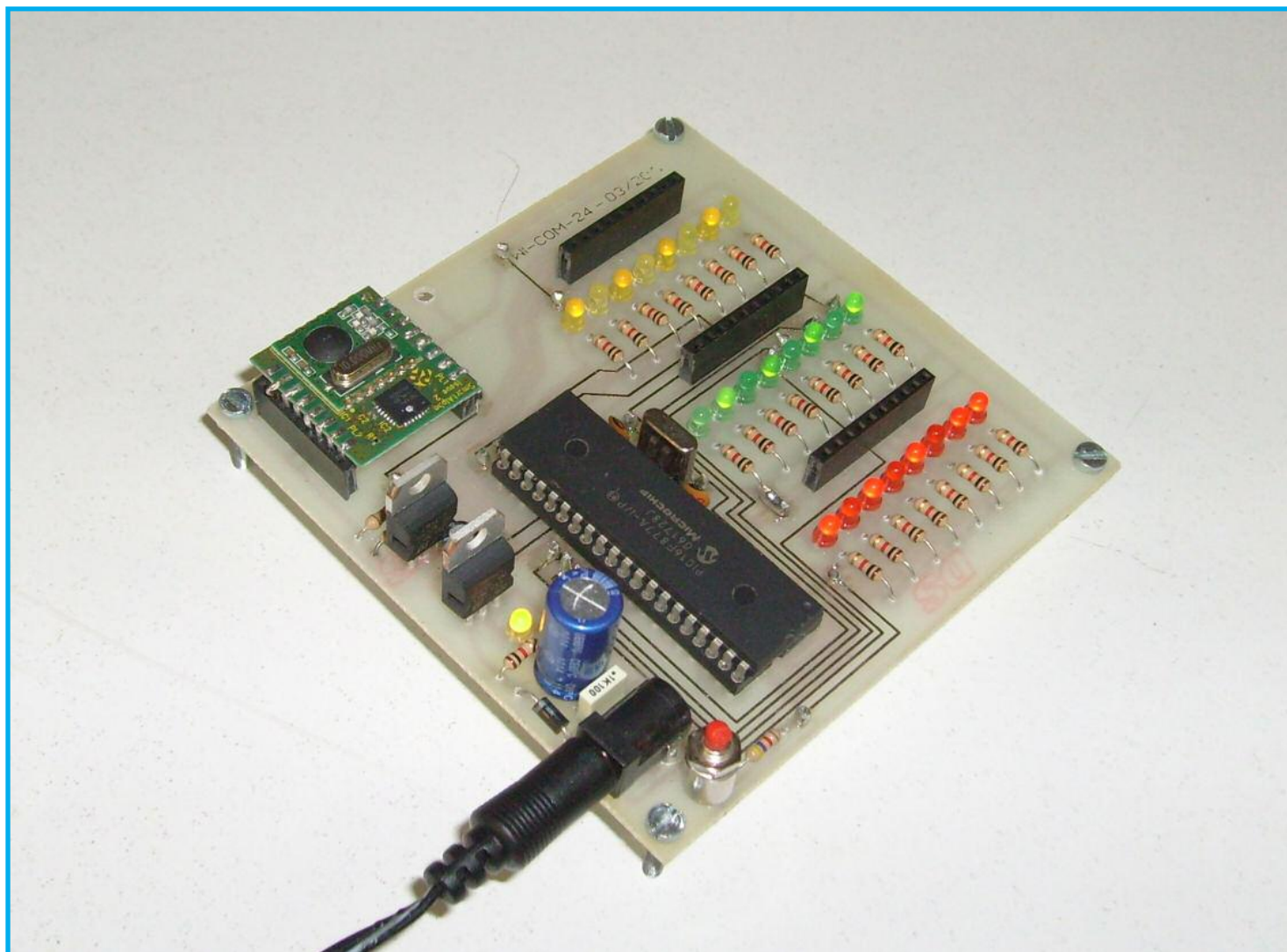



Figura 7: il circuito realizzato

In Figura7 vi è la foto del circuito realizzato. Le uscite della scheda slave sono a bassa potenza. Per poter comandare dei carichi di potenza bisogna interfacciarle con un circuito idoneo per esempio del tipo raffigurato in Figura 8.

Figura8: circuito di interfaccia

Per il master:
in Figura9 è rilevabile il pcb lato rame, in Figura10 montaggio lato rame dei componenti , in Figura11 lo schema di montaggio lato componenti
Per lo slave:
in Figura12 è rilevabile il pcb lato rame, in Figura13 il pcb lato componenti e in Figura14 lo schema di montaggio lato componenti.
Nella parte II analizzeremo il software da installare nel PC, realizzato in linguaggio Python ed il firmware per il PIC16F877A realizzato in MIKROBASIC-PRO. Infine tratteremo l'attivazione di tutto il sistema.

Figura9: circuito stampato lato rame per il master

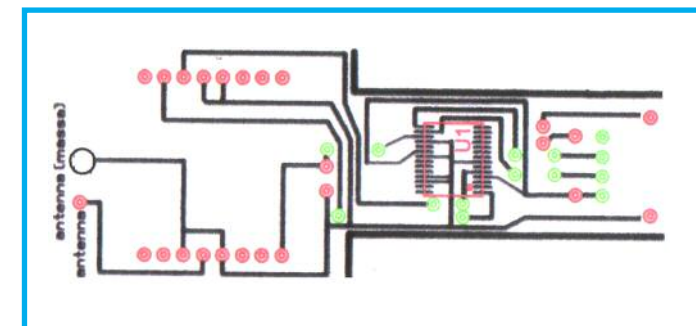
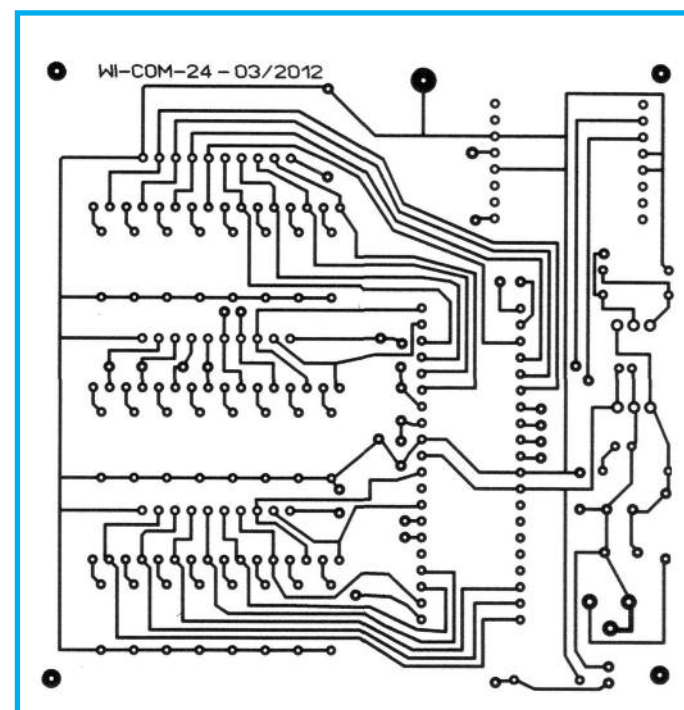



Figura12 il pcb lato rame dello slave

Figura13 il pcb lato componenti (slave)

Figura10 montaggio lato rame dei componenti (master)

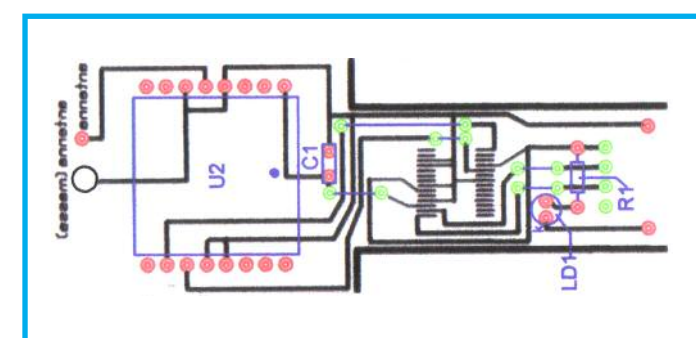



Figura11 lo schema di montaggio lato componenti (master)

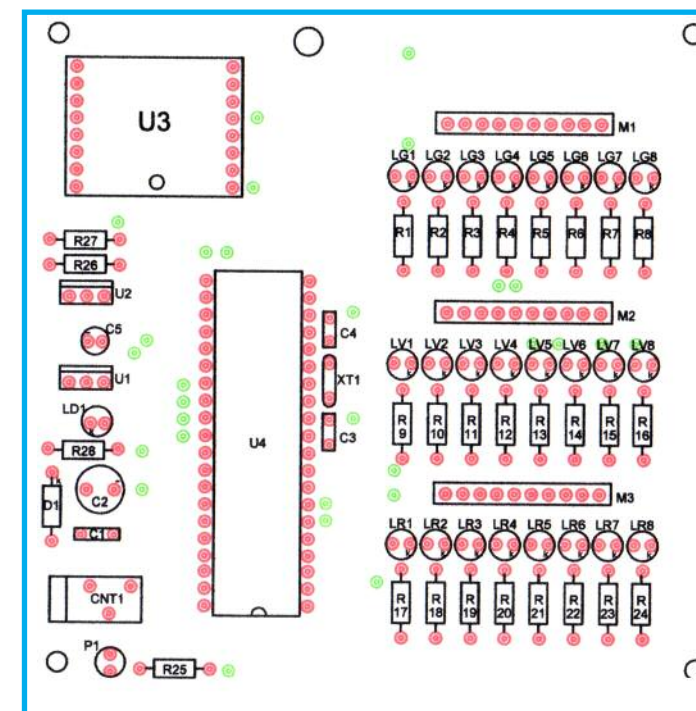
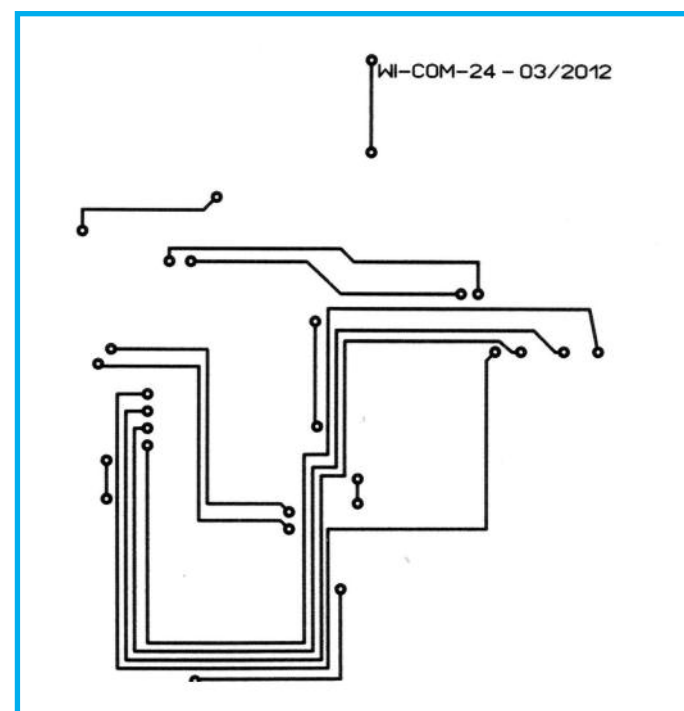



Figura14 schema di montaggio lato componenti (slave)

IEshop

VGA GRAPHICS ENGINE - PICASO

Il modulo uVGA-III è un motore grafico VGA standalone molto efficace, compatto ed economico guidato dal controller grafico PICASO. E 'in grado di fornire una soluzione grafica QVGA/VGA/WVGA a qualsiasi progetto embedded con la sua grafica potente, testi, immagini, animazioni e innumerevoli altre funzionalità integrate all'interno del modulo. Il PICASO funziona come dispositivo autonomo. Esso consente all'utente di assumere il controllo completo di tutte le risorse disponibili sulla piattaforma hardware uVGA-III, come le porte seriali, scheda di memoria uSD Card, pin di I/O, ecc. Questo elimina la necessità di un controller/processore host esterno per guidare il modulo uVGA-III tramite comandi seriali. Esso fornisce all'utente il controllo completo del modulo hardware permettendo di sviluppare rapidamente applicazioni pratiche. Programmato con 4D Systems Workshop 4 IDE Software, il uVGA-III potrebbe essere la soluzione ideale embedded di grafica VGA per il vostro progetto.

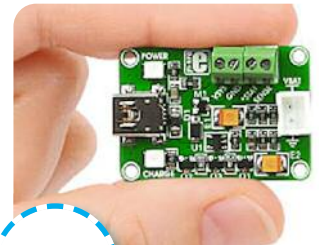
 Prezzo: € 56.87

QUADRICOTTERO IN KIT

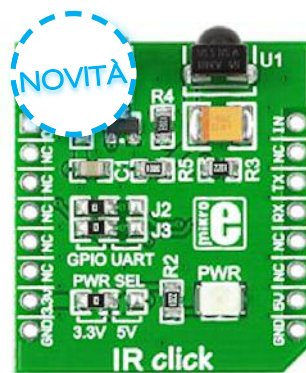
L'ELEV-8 Quadcopter (in italiano quadricottero ovvero elicottero a quattro rotori) è una piattaforma robotica volante che viene sollevata e spinta da quattro rotori fissi. Non ci sono ali fisse, tutto "il sollevamento" è creato dai rotori. A differenza di elicotteri standard, un quadcopter utilizza pale a passo fisso il cui passo rotore non varia come le pale che ruotano; il controllo del movimento del veicolo si ottiene variando la velocità relativa di ogni rotore per cambiare la spinta e la coppia prodotta da ciascuno. ELEV-8 usa una scheda HoverFly con un microprocessore multicore Propeller per controllare la stabilità del velivolo elettronicamente. I beneficio di questo sistema è una piattaforma stabile, senza collegamenti meccanici, per un piccolo velivolo agile e facilmente manovrabile. ELEV-8 è una soluzione economica per entrare nel mondo dei quadricotteri. Il kit da assemblare include: frame, hardware di montaggio, motori, controller della velocità, propulsori e la scheda di controllo per la stabilità di volo. (La sola cosa di cuiavete bisogno è l'equipaggiamento radio RC e la batteria). Raccomandiamo una radio RC a sei canali. L'ELEV-8 è abbastanza grande per il volo all'aperto e ha abbondanza di spazio per carico di allegati (fino a circa 1 kg).

 Prezzo: € 562.65

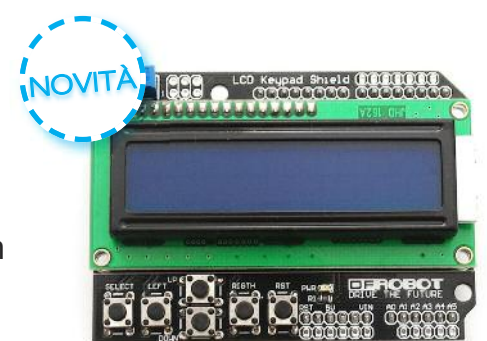
CARICABATTERIE INTELLIGENTE USB PER BATTERIE LI-POLYMER

La scheda VOLT è un caricabatterie intelligente USB per batterie Li-Polymer. È dotata di un circuito di power management con MCP73832, che può caricare la batteria oltre ad avere un connettore USB. Utilizzando i morsetti a vite forniti (VSYS e GND) la scheda VOLT può essere utilizzata anche per alimentare il dispositivo target da batteria (Vbat) o USB (5V). Quando la USB è collegata, si carica la batteria e l'uscita VSYS è a 5V. Il Power LED è acceso quando o la connessione USB o la batteria è collegata, mentre il Charge LED si illumina solo quando la batteria è in carica. Ulteriori morsetti a vite con linee STAT e SENSE consentono il monitoraggio di carica della batteria.

 Prezzo: € 13.31

CONTROLLO A RAGGI INFRAROSSI

La IR Click è una scheda accessoria che utilizza il nuovo socket mikroBUS. Si tratta di una soluzione compatta e semplice per l'aggiunta di un modulo di controllo a raggi infrarossi (IR) per il vostro progetto. È dotato di modulo ricevitore IR TSOP38338 e di un Diodo emettitore IR QEE113. Il ricevitore a frequenza portante di 38 kHz è raccomandato per codici RCMM, NEC, RC5, RC6, r-step e XMP. La IR Click comunica con il microcontrollore target attraverso i pin mikroBUS UART (Tx e Rx) o AN e linee PWM, i Jumper J2 e J3 consentono di scegliere tra questi due modi. Il jumper J1 SMD a zero-ohm viene utilizzato per selezionare tra l'alimentazione a 3.3V o alimentatore 5V. Di default è saldato in posizione 3.3V.

 Prezzo: € 9.20

LCD SHIELD PER ARDUINO

La LCD Shield for Arduino permette ad un scheda Arduino di visualizzare informazioni su un elegante display LCD con scritta bianca e retroilluminazione blu. Grazie alla presenza dei 5 pulsanti è possibile implementare progetti con menù di controllo visualizzati sul display LCD e la cui gestione avviene tramite la pressione dei 5 pulsanti, ciascuno dei quali numerato da 1 a 5. A completare la dotazione della scheda sono presenti un trimmer per la regolazione del contrasto del display ed un pulsante per il reset della scheda Arduino sottostante. La

scheda è completa dei connettori ufficiali per shield Arduino, in questo modo sarà possibile accedere ai rimanenti segnali Arduino non utilizzati da questa shield.

 Prezzo: € 18.39

UN LIBRO SUI PIC

Se volete comprendere cosa sono i microcontrollori e come essi operano, questo manuale fa per voi. Un elevato numero di illustrazioni e pratici esempi uniti a una dettagliata descrizione del PIC16F887 vi faranno lavorare ottimamente con i microcontrollori PIC. LIBRO IN LINGUA INGLESE. CONTIENE CD-ROM

 Prezzo: ~~€ 22.99~~ € 18.15

CONVERTITORE LUCE/FREQUENZA

Un convertitore luce-frequenza, l'uscita può essere un treno di impulso o un'onda quadrata (50% duty cycle) con frequenza direttamente proporzionale ad intensità della luce.

 Prezzo: ~~€ 16.99~~ € 14.52

PACCO PIC16 ASSORTITI

BUNDLE PICMICRO 16, COMPRENDE I SEGUENTI PIC (1 PZ. PER TIPO):

PIC16C505-04I/P - 14dip, 12 I/O, 1024x12 otp, 72ram, temperatura industriale
PIC16C55-JW - 28dip, 20 I/O, 512x12 eprom, 25ram, finestrato
PIC16C56-JW - 18dip, 12 I/O, 1024x12 eprom, 25ram, finestrato
PIC16C64A-JW - 40dip, 33 I/O, 2048x14 eprom, 128ram, 1pwm, finestrato
PIC16C65B-20/P - 18dip, 13 I/O, 1024x14 eprom, 68ram, 4adc, finestrato
PIC16C711-JW - 18dip, 13 I/O, 1024x14 eprom, 68ram, 4adc, finestrato
PIC16F676-I/P - Microcontrollore PIC 1024x14 byte Flash, 64 byte RAM, 128 EEPROM, 12 I/O, 8 ADC 10 bit, 1 comparatore, package 14 DIL
PIC16F84A-04/P - 18dip, 13 I/O, 1024x14 flash, 68ram
PIC16F84A-20/P - 18dip, 13 I/O, 1024x14 flash, 68 ram
PIC16C76-04I/SP - 28dip, 22 I/O, 8192x14 otp, 368ram, 5adc, 2pwm, re
PIC16C57-XTI/P - 28dip, 20 I/O, 2048x12 otp, 72ram, temperatura industriale
PIC16F877A-I/P - Microcontrollore PIC flash a 40 pin, 33 I/O, 8192x14 flash, 368 bytes ram, 256 bytes di eeprom, 8 adc a 10 bit, 2 pwm a 10 bit, frequenza 20 MHz, temperatura industriale, package DIP
PIC16C57-RC/P - 28dip, 20 I/O, 2048x12 otp, 72ram
PIC16C62B-04ISP - 28dip, 22 I/O,2048x14 otp, 128ram, 1pwm, temperatura Industriale
PIC16C72A-04/SP - 28dip, 22 I/O, 2048x14otp, 128ram, 5adc 1pwm
PIC16C73B-04ISP - 28dip, 22 I/O,4096x14 otp, 192ram, 5adc, 2pwm, temperatura industriale
PIC16C56A-04/P - 18dip, 12 I/O, 1024x12 otp, 25ram
PIC16C67-20/P - 40dip, 33 I/O, 8192x14 otp, 368ram, 2pwm
PIC16C74B-20I/P - 40dip, 33 I/O, 4096x14otp, 192ram,8adc,2pwm, temperatura industriale
PIC16C55A-JW - 28dip, 20 I/O, 512x12 eprom, 24ram, finestrato
PIC16C622A-04I/ -18dip, 13 I/O,2048x14otp, 128ram, 2comp, temperatura Industriale
PIC16C622A-JW - 18dip, 13 I/O,2048x14 otp, 128ram, 2comp, finestrato
PIC16C62A-JW - 28dip, 22 I/O, 2048x14 eprom, 128ram, 1pwm, finestrato
PIC16C63A-04/SO - 28soic, 22 I/O, 4096x14 otp,192 ram, 2pwm
PIC16C72-JW - 28dip, 22 I/O, 2048x14 eprom,128ram,5adc,1pwm, finestrato
PIC16C72A-JW - 28dip, 22 I/O, 2048x14 eprom,128ram,5adc,1pwm, finestrato
PIC16F876A-I/SP - Microcontrollore PIC flash a 28 pin, 22 I/O, 8192x14 flash, 368 bytes di ram, 5 adc a 10 bit, 2 pwm a 10 bit, frequenza 20 MHz, temperatura industriale, package DIP
PIC16C54B-04/P - 18dip, 12 I/O, 512x12ram, 25ram
PIC16C67-JW - 40dip, 33 I/O, 8192x14 eprom, 368ram, 2pwm
PIC16C76-JW - 28dip, 22 I/O, 8192x14eprom,368ram,5adc,2pwm, finestrato
PIC16F648A-I/P - Microcontrollore PIC 4096x14 flash, 224 byte RAM, 256 EEPROM, 16 I/O, 2 comparatori, 1 PWM 10 bit, package 18 DIL
PIC16C55A-04/P - 28dip, 20 I/O, 512x12 otp, 24ram
PIC16C65A-JW - 40dip, 33 I/O, 4096x14 eprom, 192ram, 2pwm, finestrato
PIC16F83-04/P - 18dip, 13 I/O, 1024x14 flash, 68ram
PIC16C57C-04/P - 28dip, 20 I/O, 2048x12 otp, 72ram
PIC16C66-04/SP - 28dip, 22 I/O, 8192x14 otp, 368ram, 2pwm
PIC16C54-JW - 18dip, 12 I/O, 512x12 eprom, 25ram, finestrato
PIC16C64A-04/P - 40dip, 33 I/O, 2048x14 otp, 128ram, 1pwm
PIC16C62A-04/SP - 28dip, 22 I/O,2048x14 otp, 128ram, 1pwm
PIC16C63A-04/SP - 28dip, 22 I/O, 4096x14 otp, 192ram, 2pwm
PIC16C72A-20ISP - 28dip, 22 I/O,2048x14otp, 128ram,5adc, 1pwm, temperatura industriale

 Prezzo: ~~€ 503.78~~ € 107.69

PACCO PIC12 ASSORTITI

BUNDLE PICMICRO 12, COMPRENDE I SEGUENTI PIC (1 PZ. PER TIPO):

PIC12CE518-04/P Microcontrollore PIC a 8 pin, 6 I/O, 512x12 otp, 16 Bytes di eeprom, 25 bytes di ram, package DIP

PIC12CE674-04I/P Microcontrollore PIC a 8 pin, 6 I/O, 2048x14 otp, 16 Bytes di eeprom, 128 bytes di ram, 4 adc a 8 bit, temperatura industriale, package DIP

PIC12C509A-JW Microcontrollore PIC a 8 pin, 6 I/O, 1024x12 eprom, 41ram, versione finestrata

PIC12CE519-JW Microcontrollore PIC a 8 pin, 6 I/O, 1024x12 otp, 16 Bytes di eeprom, 41 bytes di ram, package DIP, modello finestrato

PIC12F675-I/P Microcontrollore PIC flash a 8 pin, 6 I/O, 1024x14 flash, 64 ram, 128 eeprom, 4 adc a 10 bit, 1 comparatore, frequenza 20 MHz, temperatura industriale, package 8 dip

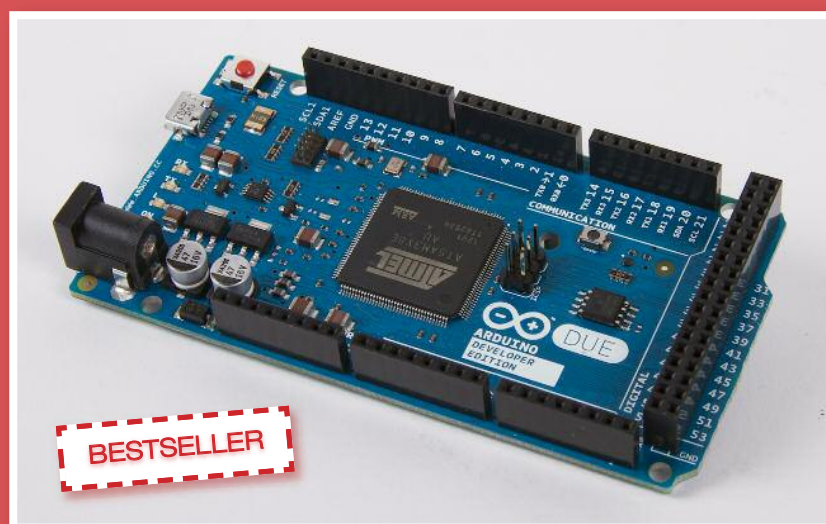
PIC12CE518-JW Microcontrollore PIC a 8 pin, 6 I/O, 512x12 otp, 16 Bytes di eeprom, 25 bytes di ram, package DIP, modello finestrato

PIC12C671-04/P Microcontrollore PIC 8dip, 6 I/O, 1024x14 otp, 128 bytes di ram, 4 adc a 8 bit

PIC12F629-I/P Microcontrollore PIC flash a 8 pin, 6 I/O, 1024x14 flash, 64 bytes di ram, 128 bytes di eeprom, 1 comparatore, frequenza 20 MHz, temperatura industriale, package 8 DIP

Prezzo: ~~€ 72.70~~ € 22.99
ARDUINO DUE

L'Arduino DUE è la nuova arrivata della famiglia Arduino. È la prima scheda basata su un microcontrollore di tipo ARM a 32bit, il SAM3X8E ARM Cortex-M3 di Atmel, che migliora le funzionalità standard di Arduino e aggiunge nuove ed interessanti funzioni.

La scheda offre 54 input/output digitali (di cui 12 possono essere usati come PWM output, con la possibilità di scegliere la risoluzione), 12 input analogici con 12bit di risoluzione, 4 UARTs (porte seriali hardware), 2 output DAC (convertitori da analogico a digitale), 84 MHz di velocità di clock, 2 connettori USB, un connettore per l'alimentazione, un connettore ICSP, un connettore JTAG e un tasto di reset. 3,3V è la massima tensione che i pin I/O possono fornire o tollerare, utilizzare una tensione maggiore, come 5V, su un pin di input può provocare il danneggiamento della scheda.
Dei due attacchi USB quello con connessione micro-USB è quello nativo ed è in grado di funzionare come una periferica USB host, il che significa che è possibile connettere alla scheda altre periferiche USB come mouse, tastiera e smartphone. 'altra porta USB, quella con connettore di tipo B, è pensate invece a scopo di debug.

Prezzo: € 47.19

MODULO DISPLAY 4,3" LCD TFT

Un modulo Display Intelligente compatto ed economico dotato di tantissime funzioni. Il cuore di questo modulo è il processore PICASO, che è guidato da un virtual core engine altamente ottimizzato: EVE (Extensible Virtual Engine). Una vasta gamma di periferiche hardware e software sono stati integrati in questo modulo, in maniera tale da dare all'utente la libertà di adattare il modulo per ogni qualsiasi applicazione.

Tra le caratteristche: display LCD TFT 4,3" con pannello touch con risoluzione di 480x272, audio, connettore card micro-SD, una porta di espansione insieme ad una serie di GPIO, pin I2C e comunicazioni seriali.

Prezzo: € 192.39

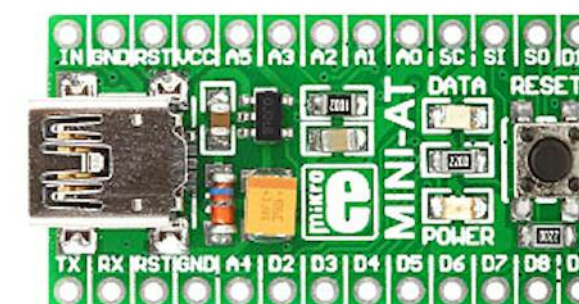
INDUSTRIAL TOUCH PANEL PC

15" INDUSTRIAL TOUCH PANEL PC BASATO SU WINDOWS EMBEDDED 7.

- Display a colori TFT 15" (1024 x 768)
- Touch Screen Resistivo
- AMD T56N 1.65GHz Dual Core APU
- 4GB DDR3 RAM
- 120GB SATA SSD
- Porta USB frontale con Copertura antipolvere e acqua

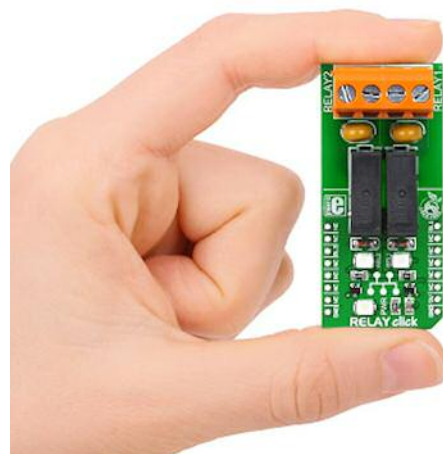
Prezzo: € 1318.90

MINI-AT BOARD

Che ci crediate o no, questa scheda è un piccolo sistema di sviluppo. Si adatta a malapena ad un dito, ma è dotato di ATmega328, USB-UART, circuito reset e indicatori LED. E' compatibile con il socket standard DIP26. Il regolatore di voltaggio on-board permette alla scheda di essere alimentata direttamente da cavo USB. E' disponibile in due versioni: 5V e 3.3V. La versione

a 5V è dotata di oscillatore a cristallo SMD a 16 MHz, mentre la versione a 3,3 V contiene un oscillatore 8MHz. Entrambi sono preprogrammati con veloce bootloader UART con 57600 bps, quindi non sono necessari programmatori esterni per lo sviluppo.

 Prezzo: € 14.72

RELAY CLICK

La scheda RELAY Click dispone di due relè di potenza sul PCB G6D1AASI-5DC, oltre a due morsetti a vite. Comunica con il microcontrollore target attraverso i pin mikroBUS PWM (RL1) e CS (RL2). Il Diodo LED (verde) indica la presenza di alimentazione. Controlla carichi DC fino a 5A, 250V AC/30V. La scheda è stato progettato per usare solo l'alimentazione a 5V, ma il livello di tensione delle linee di comunicazione possono essere nel range tra 1,8 V e 5V. I transistor on-board sono usati per pilotare i relè a corrente negativa.

 Prezzo: € 17.42

- Sei già cliente? Risparmierai il 10% sul tuo nuovo ordine!

- Il tuo ordine supera i 200 EUR? Le spese di trasporto sono Gratis!

- Vuoi ricevere un buono sconto di 5 EUR? Recensisci i prodotti acquistati!

- Vuoi ricevere un coupon del 10% di sconto? Rispondi ai quesiti di Elettroquiz!

- Vuoi ricevere particolari offerte o promozioni? Diventa membro Inware Edizioni

SE LA VIRGOLA è mobile...

EDITORIALE

Nella programmazione dei sistemi, spesso si opta per l'utilizzo di aritmetica in virgola fissa. Questa codifica prevede che un numero sia rappresentato da un numero prefissato di cifre su cui esprimere la parte intera e la parte decimale. Utilizzare l'aritmetica in virgola fissa consente di semplificare notevolmente il codice che implementa le operazioni aritmetiche, ma riduce drasticamente l'ampiezza dell'intervallo all'interno del quale un numero diviene rappresentabile. Se la virgola è mobile, si supera brillantemente questa limitazione, destinando ad esempio più cifre ai decimali qualora la parte intera sia molto piccola (e quindi in virgola fissa sarebbe preceduta da zeri). L'aritmetica in virgola mobile è però più complicata da gestire dal punto di vista della programmazione, ma i moderni tools di sviluppo consentono di superare questa limitazione rendendo la programmazione molto più semplice. E' il caso Vivado Suite di Xilinx che permette l'implementazione di hardware in virgola mobile in FPGA in maniera piuttosto semplice ed intuitiva. Scopritene le caratteristiche nell'articolo Focus On di questo mese. Magari il vostro prossimo progetto potrebbe nascere direttamente in virgola mobile.

Sistemi operativi realtime quale scegliere?

In questo articolo vedremo quali sono le ragioni che inducono alla scelta di un sistema realtime ed i criteri di scelta del giusto prodotto

Progettazione a virgola mobile con Vivaldo HLS di Xilinx

La capacità di realizzare facilmente hardware con funzione aritmetica a virgola mobile da un codice sorgente C/C++ su FPGA di Xilinx è una caratteristica potente del tool HLS Vivado. Tuttavia, la matematica a virgola mobile non è così immediata come potrebbe sembrare

Progettazione a virgola mobile con Vivado HLS di Xilinx

di JAMES HRICA, XILINX, INC.

La capacità di realizzare facilmente hardware con funzione aritmetica a virgola mobile da un codice sorgente C/C++ su FPGA di Xilinx è una caratteristica potente del tool HLS Vivado. Tuttavia, la matematica a virgola mobile non è così immediata come potrebbe sembrare

Gran parte dei progettisti usano la logica aritmetica a virgola fissa per eseguire funzioni matematiche nei propri progetti perché il metodo è rapido e molto efficiente in termini di occupazione di area. Tuttavia, ci sono molti casi in cui realizzare calcoli matematici usando un formato numerico a virgola mobile costituisce la scelta migliore. Mentre i formati a virgola fissa possono raggiungere risultati precisi, un determinato formato è caratterizzato da un intervallo dinamico molto limitato, e così i progettisti devono effettuare un'analisi profonda allo scopo di determinare gli schemi di crescita dei bit relativi ad un progetto complesso. E nel realizzare formati a virgola fissa, i progettisti devono anche introdurre molti tipi di dati intermedi (di diversi formati a virgola fissa) per ottenere una qualità ottimale dei risultati (QoR). In alternativa, i formati a virgola mobile rappresentano numeri reali in un intervallo dinamico molto più ampio, che consente ai pro-

gettisti di usare un singolo tipo di dati attraverso le lunghe sequenze di calcoli che molti algoritmi richiedono. Da un punto di vista della progettazione hardware mettere a punto un formato a virgola fissa manualmente ha un costo. Questo formato richiede un'area maggiore per essere realizzato e aumenta la latenza, dato che la logica necessaria per realizzare una determinata operazione aritmetica è considerevolmente più complessa rispetto alla logica richiesta per eseguire l'aritmetica ad interi (a virgola fissa). Fortunatamente, un nuovo tool di Xilinx, Vivado™ HLS (High Level Synthesis), aiuta i progettisti a trasformare le specifiche di progetto C/C++ in una realizzazione RTL (Register-Transfer-Level) per progetti che richiedono calcoli a virgola mobile. Vivado HLS riduce drasticamente lo sforzo di progetto che occorre per avere gli algoritmi a virgola mobile realizzati su hardware. Anche se gli elementi di base dell'esecuzione di HLS su progetti a virgola mobile sono ragionevolmente immediati, certi aspetti più sottili meritano una spiegazione dettagliata. Consideriamo alcuni degli aspetti, sia di base, sia avanzati, relativi alle prestazioni del progetto, all'area e alla verifica dell'esecuzione della logica a virgola mobile su FPGA di Xilinx® usando il tool Vivado HLS.

Details

Component

	DSP48E	FF	LUT
top_grp_fu_46_ACMP_fadd_1_U (top_grp_fu_46_ACMP_fadd_1)	2	170	269
Total	2	170	269

Expression

Figura 1 - Questo rapporto di Vivado HLS mostra un singolo core sommatore istanziato.

Summary of overall latency (clock cycles)

- Best-case latency: 132
- Average-case latency: 132
- Worst-case latency: 132

Summary of loop latency (clock cycles)

Loop 1

- Trip count: 32
- Latency: 129
- Pipeline II: 4
- Pipeline depth: 6

Figura 2 - In questo rapporto Vivado HLS, la notazione "Pipeline II = 4" indica un collo di bottiglia

DATI A VIRGOLA MOBILE E DOPPI

Il tool Vivado HLS supporta i tipi di dati C/C++ a virgola mobile e i dati doppi, che sono basati su formati binari a virgola mobile a precisione singola e doppia come definiti dello standard IEEE754.[1] PG060, la Guida Prodotti V. 6.1 dell'IP LogiCORE™ Operatore a Virgola Mobile [2] di Xilinx [2], offre analogamente un buon sommario dei formati a virgola mobile e alla realizzazione aritmetica. Una considerazione molto importante quando si progetta con ope-

razioni a virgola mobile è che questi formati numerici non possono rappresentare ogni numero reale e quindi hanno una precisione limitata. Questo punto è più sottile e complicato di quanto potrebbe sembrare a prima vista, ed è stato scritto molto su questo argomento - i progettisti sono incoraggiati a leggere accuratamente i riferimenti [3, 4, 5 e 6]. Generalmente parlando, non vi dovreste aspettare una corrispondenza esatta (a livello di rappresentazione binaria) per i risultati degli stessi calcoli effettuati da diversi algoritmi o persino diverse realizzazioni (microarchitetture) dello stesso algoritmo, anche in un contesto di puro software. Esistono diverse cause per le incongruenze di questo tipo, incluso l'accumulo di un errore di arrotondamento, che può essere sensibile all'ordine in cui le operazioni sono valutate. Inoltre, il supporto della FPU in precisione estesa può impattare sull'arrotondamento dei risultati. Con il formato x87 a 80 bit, ad esempio, le istruzioni SIMD (SSE, ecc.) si comportano diversamente dal formato x87 stesso. In più, molti valori letterali a virgola mobile possono solo essere rappresentati in modo approssimativo, anche per numeri razionali. Altri fattori che possono portare ad incongruenze includono le approssimazioni delle funzioni di libreria, ad esempio le funzioni trigonometriche, e la propagazione co-

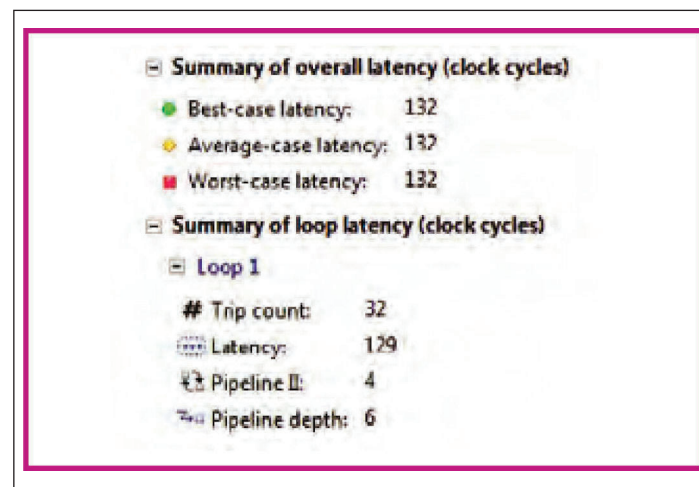



Figura 3 - In questo rapporto, Pipeline II è ancora 4, ma il numero di iterazioni scende da 32 a 8.

stante o gli effetti di ripiegamento. In aggiunta, alcune esecuzioni aritmetiche a virgola mobile supportano dati "subnormali" per rappresentare numeri più piccoli rispetto a quelli che il formato normale a virgola mobile può rappresentare. Per esempio, nel formato a precisione singola, il valore normale a virgola mobile più piccolo è 2^{-126}. Tuttavia, quando vengono supportati i dati subnormali, i bit di mantissa sono usati per rappresentare un numero a virgola fissa con un valore di esponente fisso di 2^{-127}. Consideriamo alcuni semplici esempi software che validano i risultati dei calcoli a virgola mobile. L'esempio 1 qui sotto dimostra che metodi diversi (e anche quanto appare essere lo stesso metodo) di effettuare lo stesso calcolo può portare a risposte leggermente diverse. Nel frattempo, l'Esempio 2 aiuta ad illustrare che non tutti i numeri, persino i valori completi (interi), hanno delle rappresentazioni esatte in formati binari a virgola mobile.

ESEMPIO 1: RISULTATI DIVERSI PER LO STESSO CALCOLO

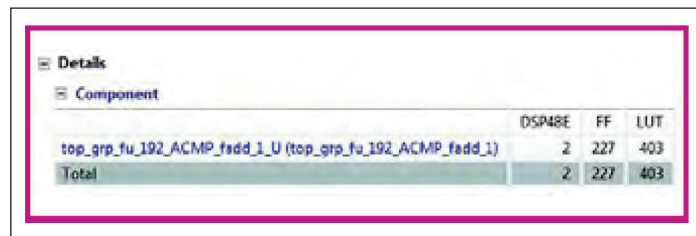
```
// Semplice demo di un problema di prevedibilità in virgola mobile
int main(void) {
float fdelta = 0.1f; // Non può essere rappresentato esattamente
float fsum = 0.0f;
while (fsum < 1.0f)
fsum += fdelta; float fprod = 10.0f * fdelta;
double dprod = float(10.0f * fdelta);
cout.precision(20);
cout << "fsum: " << fsum << endl;
cout << "fprod: " << fprod << endl;
cout << "dprod: " << dprod << endl;
return 0;
}
```

Uscita del programma:
fsum: 1.0000001192092895508
fprod: 1
dprod: 1.0000000149011611938

Il primo risultato in uscita nell'Esempio 1 è il risultato della somma dell'approssimazione di 0,1 per dieci volte, che porta all'accumulo di errori di arrotondamento. Su ciascuna iterazione, aggiungiamo il valore inesatto a precisione singola 0,1 alla somma in corso, che è quindi immagazzinata in un registro a precisione singola (a 32 bit). Al crescere dell'esponente (base -2) della somma (da -4 a 0), l'arrotondamento della somma intermedia avviene quattro volte, indipendentemente dalla precisione interna dell'unità a virgola mobile (FPU). Per il secondo valore, il calcolo è effettuato usando la precisione estesa x87 e il risultato è arrotondato prima di essere archiviato in un formato a precisione singola. Per il terzo valore, la moltiplicazione è anche effettuata con precisione estesa, ma è arrotondata e memorizzata in un formato a doppia precisione, portando ad un'accuratezza diversa del risultato. Nota: questo codice potrebbe produrre risultati diversi da quelli mostrati quando compilati per diverse architetture di macchine o con compilatori diversi.

ESEMPIO 2: ANCHE I NUMERI INTERI POSSONO PERDERE PRECISIONE

```
// Un'altra demo del problema della prevedibilità dei numeri a virgola mobile
int main(void)
{
int i;
float delta = 1.0f;
for (int i = 0; i < 100000000; i++) {
float x = (float)i + delta;
if (x / (float)i <= 1.0f) {
// (x / i) dovrebbe essere sempre > 1.0
printf("!!! ERRORE sull'iterazione #%d
```

```
!!!\n", i + 1);
return -1;
}
}
return 0;
}
```

Uscita dal programma:
!!! ERRORE sull'iterazione #16777217 !!!

Questo esempio mostra che i valori interi al di sopra di 16.777.216 (2^{24}) perdono di precisione quando sono archiviati in un formato a virgola mobile con precisione singola. Questo è perché il significando ha 24 bit e al di là di quel punto, i bit meno significativi di i devono essere scartati quando sono trasportati sul formato a virgola mobile.

Details
Component

	DSP48E	FF	LUT
top_grp_fu_192_ACMP_fadd_1_U (top_grp_fu_192_ACMP_fadd_1)	2	227	403
Total	2	227	403

Figura 4 - In questo rapporto di Vivado HLS, il sommatore di default usa due risorse DSP48E.

CONCETTI DI BASE DELLA PROGETTAZIONE A VIRGOLA FISSA USANDO IL TOOL VIVADO HLS

Il supporto nativo a HLS degli operatori aritmetici di base (+, -, *, /), gli operatori di relazione (==, !=, <, <=, >, >=) e le conversioni di formato (ad es. intero/a virgola fissa verso il formato a virgola mobile e a virgola mobile verso il formato doppio) è effettuato mappando queste operazioni sui core Operatori a Virgola Mobile LogiCORE IP di Xilinx istanziati nel risultante RTL. In aggiunta,le chiamate alla famiglia di funzioni sqrt() (dalla libreria standard di operatori matematici C/C++), il codice della forma 1.0/x e 1.0/sqrt(x), sono mappate su core Operatori a Virgola Mobile appropriati. Sebbene il CORE Generator™ possa realizzare questi core per tipi di dati personalizzati, a virgola mobile di precisione, il tool Vivado HLS genera solo le versioni a precisione singola e doppia dei core descritti dallo standard IEEE 754. Una sorgente potenziale di (piccole) differenze nei risultati generati dal software rispetto all'hardware basato sui core Operatori a Virgola Mobile e che questi core gestiscono ingressi subnormali "scaricandosi a zero", ossia, essi sono sostituiti da 0,0 quando sono incontrati. Per dettagli che riguardano il comportamento di questi core, si veda la Guida di Prodotto PG060, LogiCORE IP Operatore a Virgola Mobile. [2] Il tool Vivado HLS supporta anche molte delle altre funzioni dalla libreria matematica standard C(99)/C++, per la quale non esistono Operatori a Virgola Mobile. Per la lista completa delle funzioni, si veda UG902, Guida Utente di Vi-

vado Design Suite: Sintesi ad Alto Livello V2012.2 [7]. Gli algoritmi di approssimazione sottostanti a molte di queste funzioni sono stati selezionati ed ottimizzati per la realizzazione su FPGA di Xilinx e le loro caratteristiche di accuratezza potrebbero differire in qualche modo da quelle specificate negli standard software. Vivado HLS esegue solo le funzioni standard della libreria matematica come operazioni a virgola mobile con precisione singola e con precisione doppia in hardware. Se l'utente chiama ciascuna di queste funzioni con argomenti interi o a virgola mobile o con variabili di ritorno, il tool eseguirà delle conversioni di formato (verso o a partire dalla virgola mobile)quando necessario ed effettuerà i calcoli in virgola mobile. I progettisti possono usare tutti gli operatori a virgola mobile e le funzioni che HLS supporta in un contesto completamente in sequenza e produrre un risultato per tipo di clock, a patto che non ci sia un percorso di ritorno lungo l'operazione. Sebbene il tool Vivado HLS possa pianificare queste operazioni per un'esecuzione sequenziale poco frequente, la realizzazione potrebbe non essere la più efficiente in termini di occupazione di area, specialmente rispetto all'utilizzo flip-flop. Questo si applica anche per "/" e per sqrt(), per i quali sono disponibili core a bassa velocità.

Details

Component

	DSP48E	FF	LUT
top_grp_fu_62_ACMP_fadd_1_U (top_grp_fu_62_ACMP_fadd_1)	0	168	630
Total	0	168	630

Figura 5: - Ora il sommatore non usa risorse DSP48E, come mostra il rapporto.

COME USARE <MATH.H> IN PROGETTI BASATI SU ANSI/ISO-C

Per usare le funzioni della libreria matematica standard supportate in progetti basati su ANSI/ISO-C, avrete bisogno di includere il file di intestazione math.h in tutti i file sorgente per effettuare chiamate verso di essi.

Le funzioni di base sono intese per operare su (e di ritorno da) valori a doppia precisione - ad esempio, double sqrt(double).

Le versioni a precisione singola di gran parte delle funzioni hanno una "f" aggiunta in coda al nome della funzione, per esempio, float sqrtf(float), float sinf(float) e float ceilf(float). È importante tenere questo in mente, perché altrimenti Vivado HLS eseguirà una versione a precisione doppia molto più grande (in termini di risorse su FPGA) anche se l'argomento e le variabili di ritorno sono a precisione singola. In più, esistono conversioni di formato che usano versioni aggiuntive ed aggiungono latenza al calcolo. Un'altra considerazione quando si lavora in ambiente ANSI/ISO-C è che quando si compila e si fa girare il codice come software (incluso il lato banco di prova C durante la co-simulazione RTL), vengono usati diversi algoritmi che sono realizzati su RTL prodotti da HLS. Nel software, vengono chiamate le funzioni libc GCC; sul lato hardware, viene usato il codice della libreria matematica del tool Vivado HLS. Questo può portare ad incongruenze a livello di bit fra i due, quando entrambi i risultati potrebbero essere molto vicini alla risposta reale (in senso analitico).

ESEMPIO 3: USO NON INTENZIONALE DELLA FUNZIONA MATH A DOPPIA PRECISIONE

```
// Conseguenze non intenzionali?
#include <math.h>
float top_level(float inval)
{
// logaritmo naturale in precisione doppia
return log(inval);
}
```

Questo esempio porta ad una realizzazione RTL che converte l'ingresso in un formato a precisione doppia, calcola il logaritmo naturale in precisione doppia, quindi converte il risultato in precisione singola per l'uscita.

ESEMPIO 4: USO ESPLICITO DELLA FUNZIONE MATH IN PRECISIONE SINGOLA

```
// Assicuratevi di usare la versione corretta delle funzioni matematiche...
#include <math.h>
float top_level(float inval)
{
// logaritmo naturale in precisione singola
return logf(inval);
}
```

I progettisti possono usare tutti gli operatori in virgola mobile e le funzioni supportate da HLS in un contesto completamente in serie, le quali producono un risultato per ciclo di clock, a patto che non ci sia alcun percorso di ritorno lungo l'operazione.

Dato che viene chiamata la versione a precisione singola della funzione logaritmo, tale versione è realizzata in RTL senza necessità di conversione del formato di ingresso/uscita.

L'USO DI <CMATH> IN PROGETTI C++

Quando si progetta in ambiente C++, il modo più immediato per ottenere supporto per la libreria matematica standard è di includere il file di intestazione di sistema <cmath> in tutti i file sorgente che chiamano le sue funzioni. Questo file di intestazione fornisce versioni sovraccaricate delle funzioni di

base (a precisione doppia), da prendere come argomenti, e restituisce valori con precisione singola (a virgola mobile) nello spazio nomi standard. Per usare la versione a precisione singola, lo spazio nomi std deve essere all'interno dell'ambiente, usando l'operatore di risoluzione ambiente (::) o in alternativa importando l'intero spazio nomi con la direttiva using.

ESEMPIO 5: RISOLUZIONE ESPLICITA DELL'AMBIENTE

```
#include <cmath>
float top_level(float inval)
{
// Logaritmo Naturale In Precisione Singola
return std::log(inval);
}
```

ESEMPIO 6: ESPOSIZIONE DEI CONTENUTI DI UNO SPAZIO NOMI ALL'AMBIENTE DEL FILE

```
#include <cmath>
using namespace std;
float top_level(float inval)
{
// Logaritmo Naturale In Precisione Singola
return log(inval);
}
```

Come con l'uso di <math.h> nei progetti ANSI/ISO-C, quando si usano le funzioni da <cmath> in codice che il tool Vivado HLS deve sintetizzare, i risultati potrebbero differire fra il codice che gira come software rispetto ad una realizzazione in RTL, perché sono usati algoritmi di approssimazione diversi. Per questo motivo, l'ambiente di programmazione fornisce l'accesso agli algoritmi usati per sintetizzare l'RTL per l'uso nella modellizzazione C++. Quando si validano le modifiche verso il codice C++ destinato ad HLS ed in seguito quando si co-simula l'RTL risultante con un banco di prova basato su C++, Xilinx raccomanda che le sorgenti HLS usino le stesse chiamate alla libreria matematica e che il codice del banco di prova usi la libreria standard C++per generare i valori di riferimento. Questo fornisce un livello aggiuntivo di verifica dei modelli HLS e delle librerie di funzioni matematiche durante lo sviluppo. Per seguire questa metodologia, dovreste includere l'intestazione <his_math.h> del tool HLS Vivado solo in ciascuno dei file sorgente che devono essere sintetizzati in RTL. Per i file sorgente impiegati solo nella validazione del progetto HDL, come il programma di test e il codice di supporto, includete il file di intestazione di sistema <cmath>. La versione HLS delle funzioni nel file di intestazione <his_math.h> è parte dello spazio dei nomi hls::. Affinché le versioni HLS siano compilate per la modellizzazione e la validazione del software, usate la risoluzione ambientale hls:: per ciascuna chiamata di funzione. Notate: non è una buona idea importare lo spazio dei nomi hls:: (attraverso “using namespace hls”) quando si usano le chiamate alla libreria di funzioni matematiche standard C++, perché questo può portare ad errori di compilazione durante la progettazione HLS. L'esempio 7a illustra questo utilizzo.

ESEMPIO 7A: IL PROGRAMMA DI TEST USA LA LIBRERIA DI FUNZIONI MATEMATICHE STANDARD C++

```
// Contents of main.cpp - The C++ test bench
#include <iostream>
#include <cmath>
using namespace std;
extern float hw_cos_top(float);
int main(void)
{
int mismatches = 0;
for (int i = 0; i < 64; i++) {
float test_val = float(i) * M_PI / 64.0f;
float sw_result = cos(test_val); //float std::cos(float)
float hw_result = hw_cos_top(test_val);
if (sw_result != hw_result) {
mismatches++;
cout << "!!! Mismatch on iteration #" << i;
cout << " — Expected: " << sw_result;
```

1. ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008; IEEE-754

2. PG060, LogiCORE IP Floating-Point Operator v6.1 Product Guide

3. http://randomascii.wordpress.com/category/floating-point/

4. http://docs.oracle.com/cd/E19957-01/806 3568/ncg_goldberg.html

5. http://www.lahey.com/float.htm

6. Adam Taylor, “The Basics of FPGA Mathematics,” Xcell Journal Issue 80; http://issuu.com/xcelljournal/docs/xcell80

7. UG902, Vivado Design Suite User Guide: High-Level Synthesis v2012.2, and UG871, Vivado Design Suite Tutorial: High-Level Synthesis v2012.2

```
cout << "\t Got: " << hw_result;
cout << "\t Delta: " << hw_result - sw_result;
cout << endl;
}
}
return mismatches;
}
```

ESEMPIO 7B: IL CODICE DI PROGETTO HLS USA LA LIBRERIA HLS_MATH

```
// Contents of hw_cos.cpp
#include <hls_math.h>
float hw_cos_top(float x)
{
// hls::cos for both C++ model and RTL co-sim
return hls::cos(x);
}
```

Quando questo codice è compilato e gira come software (ad es., "Run C/C++ Project" nella GUI dell'HLS Vivado), i risultati restituiti da hw_cos_top() sono gli stessi valori di quelli che l'RTL generato da HLS produce, e il programma verifica la presenza di incongruenze rispetto al modello di riferimento software - ossia, std::cos(). Se invece avete aveste incluso <cmath> in hw_cos.cpp, non ci sarebbero incongruenze quando il progetto C/C++ era compilato e fatto girare come software, ma ci sarebbe durante la co-simulazione RTL. È importante non ipotizzare che il tool HLS Vivado effettui ottimizzazioni che sembrano ovvie e banali all'occhio umano. Come è il caso di gran parte dei compilatori software C/C++, le espressioni che implicano valori precisi in virgola mobile (costanti numeriche) potrebbero non essere ottimizzate durante HLS. Considerate il seguente codice di esempio.

ESEMPIO 8: ALGEBRICAMENTE IDENTICI; ESECUZIONI HLS MOLTO DIVERSE

```
//3  risultati diversi
void top(float *r0, float *r1, float *r2, float inval)
{
// moltiplicatore & conversioni a doppia precisione
*r0 = 0.1 * inval;
// moltiplicatore con precisione singola
*r1 = 0.1f * inval;
// divisore con precisione singola
*r2 = inval / 10.0f;
}
```

Se questa funzione è sintetizzata in RTL, si producono tre circuiti molto diversi per il calcolo di ciascuno dei valori r0, r1 ed r2. Per le regole C/C++, il valore letterale 0,1 significa un numero in precisione doppia che non può essere rappresentato esattamente. Di conseguenza, il tool istanzia un core moltiplicatore in precisione doppia (doppio), assieme ai core per convertire in-

val al doppio e il prodotto di nuovo in float (tipo r0). Quando desiderate una costante con precisione singola (in virgola mobile), aggiungete una f al valore letterale, per esempio, 0.1f. Di conseguenza, il valore r1 qui sopra è il risultato di una moltiplicazione con precisione singola fra la rappresentazione (inesatta) in virgola mobile di 0,100 e di inval. Infine, viene prodotto r2 da un core che effettua la divisione con precisione singola, con inval come numeratore e 10.0f come denominatore. Il valore reale 10 è rappresentato esattamente in formati binari in virgola mobile. Di conseguenza, in relazione al valore di inval, il calcolo r2 potrebbe essere esatto, mentre né r0 né r1 saranno probabilmente esatti. Dato che l'ordine in cui le operazioni in virgola mobile avvengono impatta potenzialmente sul risultato (ad esempio, a causa degli arrotondamenti che avvengono più volte), più valori esatti in virgola mobile inclusi in un'espressione potrebbero non essere ripiegati assieme.

ESEMPIO 9: L'ORDINE DELLE OPERAZIONI PUO' IMPATTARE IL RIPIEGAMENTO DI COSTANTI

```
// realizzazioni molto diverse
void top(float *r0, float *r1, float inval)
{
// *r0 = inval; costanti eliminate
*r0 = 0.1f * 10.0f * inval;
// due moltiplicazioni con precisione
doppia
*r1 = 0.1f * inval * 10.0f;
}
```

In questo esempio, per via dell'ordine di valutazione dell'espressione assegnata ad r0, il compilatore riconosce l'intera espressione come essere un'identità e non genera alcun hardware. Tuttavia, lo stesso non si applica ad r1; vengono effettuate due moltiplicazioni.

ESEMPIO 10: EVITARE VALORI PRECISI IN ESPRESSIONI DI INTERI

```
void top(int *r0, int *r1, int inval)
{
*r0 = 0.5 * inval;
*r1 = inval / 2;
}
```

Per questo esempio, HLS realizza la logica per assegnare r0 convertendo inval al formato con precisione doppia allo scopo di moltiplicarlo per 0,5 (un valore esatto in precisione doppia), e quindi lo converte di nuovo in un intero. D'altro canto, HLS moltiplica la moltiplicazione e la divisione per potenze intere di due rispettivamente in operazioni a scorrimento a sinistra e a destra, e li realizza in hardware come semplici selezioni di fili (con zero piazzole o con l'estensione del segno come appropriato per la direzione e il tipo dell'operando). Di conseguenza, la logica creata per assegnare r1 è molto più efficiente raggiungendo al contempo lo stesso risultato aritmetico.

PARALLELISMO, CONCORRENZA E CONDIVISIONE DELLE RISORSE

Dato che le operazioni in virgola mobile usano considerevoli risorse relative ad operazioni su interi o a virgola fissa, il tool HLS Vivado utilizza queste risorse nel modo il più possibile efficiente. Il tool condividerà spesso core Operatori in Virgola Mobile fra più chiamate all'operazione sorgente quando le dipendenze dei dati e i vincoli lo permettono. Per illustrare questo concetto, sommiamo quattro valori in virgola mobile nell'esempio che segue.

Dato che le operazioni in virgola mobile usano risorse considerevoli relativamente alle operazioni su interi o a virgola fissa, il tool HLS Vivado utilizza tali risorse nel modo il più possibile efficiente.

ESEMPIO 11: PIU OPERAZIONI USANO UN SINGOLO CORE

```
// Quanti core sommatori?
void top (float *r, float a, float b, float c,
float d)
{
*r = a + b + c + d;
}
```

Talvolta, quando l'ampiezza della banda dati lo consente, potrebbe essere desiderabile effettuare più lavoro in un determinato tempo effettuando contemporaneamente numerose operazioni, che altrimenti sarebbero pianificate in modo sequenziale. Nell'esempio che segue, l'RTL che il tool ha creato genera i valori nella matrice dei risultati sommando gli elementi di due matrici sorgente in un anello in sequenza. Vivado HLS mappa gli argomenti della matrice di livello superiore nelle interfacce di memoria, limitando così il numero di accessi per ciclo (ad esempio, due per ciclo per RAM a due porte, una per ciclo per le FIFO, ecc.).

ESEMPIO 12: SOMME INDIPENDENTI

```
// somme indipendenti, ma l'I/O consente solo
//l'emissione di un risultato per ciclo
void top (float r0[32], float a[32], float
b[32])
{
#pragma HLS interface ap_fifo
port=a,b,r0
for (int i = 0; i < 32; i++) {
#pragma HLS pipeline
r0[i] = a[i] + b[i];
}
}
```

Per default, il tool HLS Vivado pianifica questo anello per irterare 32 volte e per realizzare un singolo core sommatore, a

patto che i dati in ingresso siano disponibili continuamente e la FIFO in uscita non si riempia mai completamente. Il blocco RTL risultante richiede 32 cicli, più alcuni per svuotare la pipeline del sommatore. Questo è essenzialmente il più possibile veloce, data la velocità dati I/O. Se d'altra parte le velocità dati vengono aumentate, allora i progettisti possono usare le tecniche HLS anche per aumentare la velocità di elaborazione. Estendendo l'esempio precedente, potete aumentare la banda I/O raddoppiando l'ampiezza delle interfacce usando la direttiva che rimodella la matrice del tool Vivado HLS. Per aumentare la velocità di elaborazione, svolgete parzialmente l'anello di un fattore due per adattarsi all'aumento di banda.

ESEMPIO 13: SOMME INDIPENDENTI

```
// Somme indipendenti, con banda I/O
// aumentata -> alta velocità e area
void top (float r0[32], float a[32], float b[32])
{
#pragma HLS interface ap_fifo port=a,b,r0
#pragma HLS array_reshape cyclic factor=2 \
variable=a,b,r0
for (int i = 0; i < 32; i++) {
#pragma HLS pipeline
#pragma HLS unroll factor=2
r0[i] = a[i] + b[i];
}
}
```

Con queste direttive aggiunte, il tool HLS Vivado sintetizza l'RTL che ha due serie di sommatori che operano in modo concorrente su oltre metà delle iterazioni per produrre due campioni in uscita per ciascuna iterazione. Questo è possibile perché ciascun calcolo è completamente indipendente e l'ordine con cui le operazioni di somma avvengono possono impattare sull'accuratezza dei risultati. Tuttavia, quando avvengono calcoli più complessi, magari come risultato di una catena di operazioni dipendenti in virgola mobile, , il tool HLS Vivado non può risistemare l'ordine di tali operazioni. Il risultato può essere meno concorrenza o condivisione di quanto atteso. In più, quando c'è retroazione o ricorrenza in un percorso dati in sequenza, l'aumento della velocità del progetto lungo l'operazione concorrente potrebbe richiedere alcune modifiche manuali del codice sorgente. L'esempio 14 presenta un esempio dettagliato di come il tool Vivado HLS si comporta con la retroazione e con la ricorrenza attraverso operazioni in virgola mobile. Il seguente codice richiede un accumulo in virgola mobile in una regione caratterizzata da esecuzione seriale.

ESEMPIO 14: DEPENDENCY THROUGH AN OPERATION

```
// Floating-point accumulator
float top(float x[32])
{
#pragma HLS interface ap_fifo port=x
float acc = 0;
for (int i = 0; i < 32; i++) {
#pragma HLS pipeline
acc += x[i];
}
return acc;
}
```

Dato che questa formula di accumulo produce una ricorrenza e la latenza per la somma in virgola mobile è generalmente più grande di un ciclo, questa sequenza non può ottenere una velocità di un accumulo per ciclo.

Ad esempio, se il sommatore in virgola mobile ha una latenza di quattro cicli, l'intervallo di iniziazione della sequenza è anch'esso di quattro cicli (come si vede nel rapporto di sintesi del tool Vivado HLS mostrato in Figura 2), a causa della dipendenza la quale richiede che ciascun accumulo si completi prima che un altro possa iniziare. Di conseguenza, la velocità migliore ottenibile in questo esempio è di un accumulo ogni quattro cicli. L'anello di accumulo itera 32 volte, richiedendo quattro cicli per volta, portando ad un totale di 128 cicli più alcuni per scaricare la sequenza. Un'alternativa a prestazioni superiori potrebbe consistere nell'interlacciare quattro accumuli parziali sullo stesso sullo stesso core sommatore, ciascuno che completa ogni quattro cicli, riducendo di conseguenza il tempo che occorre per effettuare 32 operazioni di somma. Tuttavia, il tool Vivado HLS non può dedurre una simile ottimizzazione dal codice fornito nell'Esempio 14, perché richiede una modifica all'ordine delle operazioni per l'accumulo. Se ciascun accumulo parziale richiede un elemento ogni quattro di x[] come ingresso, l'ordine delle somme individuali cambia, il che potrebbe portare ad un risultato diverso. Potete lavorare attorno a questa limitazione effettuando piccole modifiche al codice sorgente per rendere il vostro intento più esplicito. Il codice di esempio che segue introduce una matrice, acc_part[4], per memorizzare le somme parziali; queste sono successivamente sommate, e l'anello principale di accumulo è parzialmente svolto.

ESEMPIO 15: RIORDINO ESPLICITO DELLE OPERAZIONI PER OTTENERE PRESTAZIONI MIGLIORI

```
// Accumulatore in virgola mobile
float top(float x[32])
{
#pragma HLS interface ap_fifo port=x
float acc_part[4] = {0.0f, 0.0f, 0.0f, 0.0f};
```

```
// Manually unroll by 4
for (int i = 0; i < 32; i += 4) {
// Partial accumulations
for (int j = 0; j < 4; j++) {
#pragma HLS pipeline
acc_part[j] += x[i + j];
}
// Accumulo finale
for (int i = 1; i < 4; i++) {
#pragma HLS unroll
acc_part[0] += acc_part[i];
}
}
return acc_part[0];
}
```

Con questa struttura di codice, il tool Vivado HLS riconosce di poter pianificare i quattro accumulatori parziali in un singolo core sommatore su cicli alternati, che è un uso più efficiente delle risorse (si veda la Figura 3). Il successivo accumulo finale potrebbe anche usare lo stesso core sommatore, in relazione ad altri fattori. Ora l'anello principale richiede otto iterazioni (32/4), ciascuna delle quali necessita di quattro cicli per produrre i quattro accumuli parziali. La stessa quantità di lavoro ha luogo in molto meno tempo con un piccolo aumento nelle risorse FPGA. L'anello finale di accumulo, pur usando lo stesso core sommatore, aggiunge ulteriori cicli, ma il numero è fissato e piccolo rispetto ai risparmi ottenuti svolgendo l'anello di accumulo principale, specialmente quando l'insieme di dati è grande. Potremmo ottimizzare ulteriormente questo passaggio finale di accumulo ma con ritorni in diminuzione relativamente al rapporto fra prestazioni e area. Quando è disponibile una banda I/O superiore, possiamo specificare fattori di svolgimento più grandi allo scopo di portare più core aritmetici da supportare. Se nell'esempio precedente, fossero disponibili due elementi x[]per ciclo di clock, potremmo aumentare il fattore di svolgimento fino a 8, in tal caso due core sommatori sarebbero realizzati e otto accumuli parziali sarebbero effettuati per ciascun ciclo. La scelta del dispositivo di riferimento e li vincoli di temporizzazione dell'utente possono avere un impatto sulla latenza esatta dell'operatore.

Generalmente, è necessario far girare HLS ed effettuare alcune analisi di prestazioni di un caso di base più semplice (ad es. l'Esempio 14) per determinare la quantità ottimale di svolgimento.

COME CONTROLLARE LE RISORSE DI REALIZZAZIONE

I core Operatori a Virgola Mobile LogiCORE IP consentono il controllo sull'utilizzo di DSP48 di alcune delle operazioni supportate. Ad esempio, il core moltiplicatore ha quattro varianti che effettuano compromessi fra le risorse di

logica (LUT) e l'uso di DSP48. Normalmente, il tool Vivado HLS determina automaticamente quale tipo di core usare in base ai vincoli di prestazioni. I progettisti possono usare la direttiva RESOURCE del tool Vivado HLS per disabilitare la selezione automatica e specificare quale tipo di core Operatori in Virgola Mobile usare per una determinata istanza di un'operazione. Ad esempio, il codice presentato nell'Esempio 14, il sommatore sarebbe generalmente realizzato avvalendosi del core "ad utilizzo completo" usando due risorse DSP48E1 su un FPGA Kintex™-7, come mostrato nella sezione "Componente" del rapporto di sintesi (si veda la Figura 4). Il codice di esempio che segue forza l'operazione di addizione ad essere mappata su un core FAddSub_nodsp, come mostrato in Figura 5.

ESEMPIO 16: COME USARE LA DIRETTIVA 'RISORSA' PER SPECIFICARE UNA VARIANTE AI CORE OPERATORI A VIRGOLA MOBILE

```
// Accumulatore in virgola mobile
float top(float x[32])
{
#pragma HLS interface ap_fifo port=x
float acc = 0;
for (int i = 0; i < 32; i++) {
#pragma HLS pipeline
#pragma HLS resource variable=acc \
core=FAddSub_nodsp
acc += x[i];
}
return acc;
}
```

Si veda UG902, la guida utente di Vivado HLS [7], per avere dettagli completi sull'uso della direttiva RESOURCE e anche per una lista dei core disponibili.

COME VALIDARE I RISULTATI DEI CALCOILI IN VIRGOLA MOBILE

Esistono molti motivi per attendersi incongruenze a livello di bit (o superiori) fra i risultati in virgola mobile dello stesso calcolo effettuato con mezzi diversi. Le incongruenze possono verificarsi in un numero di posti, inclusi diversi algoritmi di approssimazione, il riordino di operazioni che portano a differenze negli arrotondamenti e nella gestione di casi subnormali (in cui i core Operatori in Virgola Mobile si scaricano a zero). In generale, il risultato della confronto di due valori in virgola mobile, specialmente per l'uguaglianza, può essere fuorviante. I due valori confrontati potrebbero differire solo di una "unità all'ultimo posto" (ULP; il bit meno significativo nel loro formato binario), che possono rappresentare un errore relativo estremamente basso, tuttavia l'operatore "==" restituisce falso. Ad esempio, usando il formato in virgola mobile con precisione singola, se entrambi gli operandi sono valori diversi da zero (e non subnormali), una differenza di 1 ULP rappresenta un errore relativo sull'ordine dello 0,00001 per cento. Per questo motivo, è una buona prassi evitare di usare gli operatori "==" e "!=" per confrontare numeri in virgola mobile. Per contro, controllate che i valori sono "abbastanza vicini", magari introducendo una soglia di errore accettabile. Per la gran parte dei casi, impostare un ULP o un livello di errore relativo accettabile funziona bene ed è preferibile alle soglie di errore assoluto (o "epsilon"). Tuttavia, quando uno dei valori confrontati è esattamente zero (0.0), questo metodo crolla. Se uno dei valori che state confrontando ha o può assumere un valore costante pari a zero, allora potreste usare invece una soglia di errore assoluto.

Il codice di esempio che segue presenta un metodo che potete usare per confrontare due numeri in virgola mobile per l'uguaglianza approssimativa, consentendovi di impostare sia ULP sia i limiti dell'errore assoluto. Questa funzione è intesa per l'uso nel codice C/C++ "banco di prova" per validare le modifiche al codice sorgente HLS e per verificarlo durante la co-simulazione RTL del tool Vivado HLS. Potete anche usare delle tecniche simili anche in codice destinato per la realizzazione HLS.

ESEMPIO 17: CODICE C PER VERIFICARE I VALORI IN VIRGOLA MOBILE PER L'UGUAGLIANZA APPROSSIMATIVA

```
// Create un tipo basato sull'unione per
il semplice accesso alla
// rappresentazione binaria
typedef union {
float fval;
unsigned int rawbits;
} float_union_t;
bool approx_eqf(
float x, float y,
int ulp_err_lim, float abs_err_lim
)
{
float_union_t lx, ly;
lx.fval = x;
ly.fval = y;
// Il confronto basato sull'ULP è probabile che sia senza significato
// quando x o y è esattamente zero o i loro segni
// differiscono, quindi la verifica rispetto una soglia di
// valore assoluto gestisce anche (-0.0 == +0.0),
// che dovrebbe restituire vero. N.B. che
// abs_err_lim deve essere scelto saggiamente, in base alla
// conoscenza di calcoli/algoritmi che guidano
// il confronto. Non c'è sostituito per
// un'adeguata analisi degli errori quan-
```

do l'accuratezza dei risultati
// conta.
if (((x == 0.0f) ^ (y == 0.0f)) ||
(__signbit(x) != __signbit(y))) {
return fabs(x - y) <= fabs(abs_err_lim);
}
// Effettuate il confronto ULP per tutti gli altri casi
return abs((int)lx.rawbits -
(int)ly.rawbits) <= ulp_err_lim;
}
Non esiste un'unica risposta in merito a quale livello impostare l'ULP e le soglie dell'errore assoluto, dato che le impostazioni varieranno in relazione al progetto. Un algoritmo complesso può avere il potenziale di accumulare molti ULP di inaccuratezza nelle proprie uscite, in base al risultato del riferimento. Altri operatori relazionali possono anche essere inclini a risultati fuorvianti. Ad esempio, quando si verifica se un valore è inferiore (o superiore) rispetto ad un altro e la differenza è di solo qualche ULP, è ragionevole risolvere il confronto in questo caso? Potremmo usare la funzione presentata prima in combinazione con un confronto inferiore a/superiore a, per contrassegnare i risultati che potrebbero essere ambigui.

UNA CARATTERISTICA POTENTE

La capacità di realizzare in modo semplice l'hardware con funzione aritmetica in virgola mobile (codice RTL) a partire da un codice sorgente C/C++ su FPGA di Xilinx è una caratteristica potente del tool Vivado HLS. Tuttavia, l'uso della matematica a virgola mobile non è così diretto come potrebbe sembrare, sia esso da un punto di vista del software, dell''hardware o di entrambi. La natura imprecisa dei formati binari in virgola mobile conformi allo standard IEEE-754 possono rendere difficile la validazione dei risultati calcolati. In più, i progettisti devono prendere ulteriori precauzioni, sia a livello del codice C/C++, sia quando si applicano le direttive per l'ottimizzazione HLS, per ottenere la QoR desiderata, sia rispetto all'utilizzo delle risorse FPGA, sia rispetto alle prestazioni del progetto.

Per maggiori consigli pratici, guardate la pagina di prodotto all'indirizzo http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/hls/; I Tutorial Video Vivado all'indirizzo http://www.xilinx.com/training/vivado/ e la pagina di prodotto "Operatore a virgola mobile" all'indirizzo http://www.xilinx.com/products/intellectualproperty/FLOATING_PT.htm

Codice MIP 2837486